Generic Reduction, and Work with Partial Computations and Partial Oracles

by

Gregory Igusa

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Mathematics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Theodore Slaman, Chair
Professor Leo Harrington
Associate Professor Sherrilyn Roush

Spring 2013

Generic Reduction, and Work with Partial Computations and Partial Oracles

Abstract

Generic Reduction, and Work with Partial Computations and Partial Oracles

by

Gregory Igusa

Doctor of Philosophy in Mathematics

University of California, Berkeley

Professor Theodore Slaman, Chair

A generic computation of a subset A of the natural numbers $\mathbb{N}$ consists of a computation which correctly computes *most* of the bits of A and never incorrectly computes any bits of A, but which does not necessarily give an answer for every input. This concept derives its motivation from a recent trend in complexity theory to study the complexity of a problem in terms of how difficult it is to compute the majority of size n instances of the problem, and not in terms of how difficult it is to compute the most difficult size n instances of the problem.

When considered from a purely recursion-theoretic point of view, generic computability proves to be somewhat counterintuitive. It can be shown to be, in some sense, as nontransitive as possible, and not only do minimal pairs not exist in this context, but even finite minimal sets of reals can be shown to not exist.

If we modify the definition of generic computation to make it transitive, we are naturally confronted with a deduction procedure in which the oracles, like the computations, are not required to be total. For this reason, "generic reduction" is defined in an inherently Π^1_1 manner. While studying generic reduction, we show that it is Π^1_1-complete, and we also present a characterization of the hyperarithmetic reals in terms of generic reduction.

Finally, we increase the level of abstraction, studying transitive and nontransitive deduction procedures with partial oracles. We provide a framework for discussing such reductions, we mention how various known results in recursion theory can be fit into this framework, and we discuss a number of seemingly trivial reductions, showing how subtle variations in the definitions can impact the produced reduction.

To Wondra Waizenegger, and Andre Kornell

For the defining impact you both had on making my years as a graduate student the best years of my life.

Contents

Acknowledgments

I want to thank my advisor Ted Slaman for his insight into the meanings behind mathematical results. His commentary on mathematics is invariably enlightening, both in terms of the intuitive reasons behind why proofs and constructions work, and also in terms of the aspects of constructions that necessarily cannot be changed. Ted's ability to instill interest in the subjects that he talks about is exceptional, and I am constantly impressed with the skill with which he keeps track of the mathematical interests, techniques, and progress of his many students.

Thanks, Ted!

Chapter 1

Generic Computation

1.1 Introduction

Background

In complexity theory, there is a frequently observed phenomenon that a given sort of question can be proved to be very difficult in the worst-case scenario, while still being very easy to solve in practice. The first extensively studied example of this phenomenon is the simplex algorithm for linear programming. A linear programming problem is a problem of maximizing a linear function on a convex polytope. The simplex algorithm, introduced by Dantzig [1] in 1947, is a sort of greedy algorithm, which uses local information to travel along the edges of the polytope until it maximizes the value of the function. It is used hundreds of times per day, always producing correct results very quickly. In 1969, Klee and Minty [11] showed that there exist linear programming problems for which the simplex algorithm takes exponential time. Then, in 1979, Khachiyan [4] showed that the linear programming problem can be solved in polynomial time. In practice, however, Dantzig's simplex algorithm runs much more quickly than Khachiyan's ellipsoid algorithm.

There is a joke told in some circles about the difference between theory and practice: in theory, there is no difference, but in practice, they are completely different. The origin and intended context of this joke are open to debate, but the joke retains a very distinct humor value in the mathematical community. This humor, perhaps, betrays a certain bias in the community: ideally, when theory and practice do not match up, theory should be modified to fit the observed world; however, in practice, it is all too frequent that these sorts of phenomena do not fit nicely into the established theoretical framework, and thus they are, in some sense, unfit for theoretical study.

Ideally, however, when a difference between theory and practice arises, a new theoretical framework is developed for exploring this difference.

In 1986, Levin [12] introduced the notion of *average-case complexity* to provide a theoretical framework for studying problems which are easier in practice than they are in theory. He notes that some NP problems, such as factoring large numbers, are difficult (or at least believed to be difficult), whereas other problems, including many NP-complete problems, are very easy to solve in the vast majority of cases. In particular, he observes that this latter sort of NP-complete problem is almost useless for cryptographic purposes.

The idea behind average-case complexity is relatively straightforward: rather than looking at the longest running time of an algorithm on a size n instance of the problem, Levin examines the expected value of the running time of the algorithm on a randomly chosen size n instance of the problem (with some slight modifications to make average-case complexity behave better with respect to compositions of functions.) This provides a means of rigorously distinguishing between questions which are actually difficult, and questions which are generally easy, but which can occasionally have difficult questions coded into them.

In 2003, Kapovich, Miasnikov, Schupp and Shpilrain [9] introduced the notion of *generic-case complexity* as a means of studying this discrepancy from a different viewpoint.

The generic-case complexity of a problem is the complexity of the majority of the instances of the problem (density-1 many instances of the problem, as will be defined in Definition 1.1.1), without any regard to the complexity of the problem in the few outlier cases. For this reason, while average-case complexity gives a slightly better idea of the overall difficulty of a problem, generic-case complexity more directly allows us to study the *difference* between the worst case complexity of a question, and the most frequently encountered complexity of that question.

One of the benefits of studying generic-case complexity is that the generic-case complexity of a problem can sometimes be computed even while the worst case complexity of the problem is unknown [9]. Indeed, there are situations where the generic-case complexity of a problem is known, while the problem is currently not known to be solvable [7, 8], or even while the problem is known to be unsolvable [9]. Currently, the majority of questions that have been analyzed from the point of view of generic-case complexity are questions regarding groups, especially regarding the word problem for groups: in a group that is presented in terms of its generators and relations, how difficult is it to determine whether or not two given words are equal? The word problem for finitely presented groups is known to not be decidable, but for every known group with undecidable word problem, the word problem is generically computable.

For our purposes, perhaps the largest benefit of generic-case complexity over average-case complexity is that generic-case complexity lends itself much more naturally to a recursion-theoretic analogue: generic computability. It is reasonable to ask which things can "usually" be computed, but it does not make much sense to ask which things can be computed "on average." Indeed, every instance of an undecidable question whose generic-case complexity has been found is a natural example of a nonrecursive set which is generically computable!

Definitions

In this paper, we follow the general notational heuristics of recursion theory. Those familiar with the subject are free to skip to Definition 1.1.1. Those unfamiliar with the subject may look to [14] if they desire a more thorough introduction than that which is provided here.

A *partial recursive funcion* is any deterministic algorithm, which could theoretically be implemented on a computer, with the property that given a natural number as input, the algorithm runs for some (possibly infinite) amount of time, and then, if it halts in a finite amount of time, it gives an output which is a natural number. Given a partial recursive function φ, if φ halts on input n, giving output m, we say that $\varphi(n)$ *converges*, denoted $\varphi(n)\downarrow$, and that $\varphi(n) = m$. If φ runs infinitely on input n, we say that $\varphi(n)$ *diverges*, or $\varphi(n)\uparrow$. The *domain* of a partial recursive function is the set of inputs on which it halts. A partial recursive function is *total* if its domain is $\mathbb{N}$. There is a natural listing of the partial recursive functions, and φ_e is the eth function on that list.

A subset of the natural numbers is known as a *real*. A real is *recursive* (alternatively, *computable*) if there is a partial recursive function φ, whose domain is all natural numbers, such that $\varphi(n) = 1$ if $n \in A$, and $\varphi(n) = 0$ if $n \notin A$, or in other words, if the characteristic function of the real is a total recursive function. A real is *recursively enumerable* (alternatively, *r.e.* or *enumerable*) if it is the domain of some partial recursive function. The domain of φ_e is denoted W_e.

A *Turing functional*, φ is a generalization of a partial recursive function which is designed to be able to work with a real A as an *oracle* in the following sense. While φ^A is running, it may at any time ask a question of the form "Is $n \in A$?" at which point it will be given the answer to that question. It may ask as many questions of this form as it wants, as often as it wants, and for as many natural numbers n as it wants. We say that A *computes* B (alternatively, B *is recursive in* A, or $A \geq_T B$) if there exists a Turing functional φ such that φ^A is a computation of B. Likewise, we say that B is *enumerable in* A if φ^A is an enumeration of B. We may assume that the eth partial recursive function, φ_e, is equal to the eth Turing functional running on the empty oracle, φ_e^0.

(There are other, perhaps more elegant, ways to define oracle computations, particularly by having φ query initial segments of A rather than querying individual elements of A, but this method is more in line with colloquial notations, and it will also be more useful to us in Chapters 2 and 3, because it will generalize better in those contexts.)

The *Turing degrees* are the equivalence classes of reals under mutual relative computability ($A \equiv_T B$ iff $A \geq_T B$ and $B \geq_T A$). If A and B are reals, the *join of* A *and* B, denoted $A \oplus B$ is the real given by $(2n \in A \oplus B \leftrightarrow n \in A)$ and $(2n + 1 \in A \oplus B \leftrightarrow n \in B)$. The most relevant thing about $A \oplus B$ is that $A \oplus B \geq_T A$, and $A \oplus B \geq_T B$, and in fact that it is the minimal upper bound of A and B in the Turing degrees.

Sometimes, for the sake of brevity, a real A is confused with its characteristic function. Likewise, a function, especially a Turing functional with an oracle, φ^A, is sometimes confused with the real computed by φ^A. (So, for example, if we write "$A(n) = B(n)$," we mean "either $n \in A$ and $n \in B$, or $n \notin A$ and $n \notin B$." Likewise, if we write "$\varphi^A = B$", we mean "A computes B via the functional φ.") A real is also frequently thought of as an infinite sequence of zeroes and ones, and the nth bit of the real is the nth number in that sequence.

The language of first order arithmetic is the first order language with constants $0, 1$, binary functions $+, \times$, and the binary predicate $<$. A formula $\Phi(\bar{x})$, of first order arithmetic, potentially with free variables, is Σ_0^0 if it can be written without unbounded quantifiers. (A quantifier is bounded if it is of the form $\exists x < y$ or $\forall x < y$.) A formula $\Phi(\bar{x})$ is Σ^0_{n+1} if it is of the form $\exists y \Psi(\bar{x}, y)$ where $\Psi(\bar{x}, y)$ is a Π^0_n formula (a formula that is the negation of a Σ^0_n formula). A real A is Σ^0_n if it can be defined by a Σ^0_n formula, $\Phi(x)$, or in other words, if $x \in A \leftrightarrow \Phi(x)$ (with all symbols interpreted in the usual way in $\mathbb{N}$). A real is Δ^0_n if it is both Σ^0_n and Π^0_n. A real, A, is *arithmetic* if there exists an n such that A is Σ^0_n.

The *jump* of a real, A, denoted A', is the real given by $A' = \{n \mid \phi^A_n(n)\downarrow\}$. A real is Δ^0_2 if and only if it is recursive in $0'$. This is true if and only if it can be written as a uniform limit of recursive sets. (More precisely, X is Δ^0_2 if and only if there is a total recursive φ such that for every n, $\lim_{s\to\infty}\varphi(\langle n, s\rangle)$ exists, and is equal to $X(n)$.) Similarly, the reals recursive in A' are precisely the reals that can be expressed as limits of functions that are recursive in A. One important fact about the jump operator is that for every A, A' is not recursive in A. If a real, B, can be defined by a sequence of n alternating quantifiers, followed by something that is recursive in A, then $B \leq_T A^{(n)}$, the nth jump of A.

Sometimes, for the sake of brevity, a real A is confused with its characteristic function. Likewise, a function, especially a Turing functional with an oracle, φ^A, is sometimes confused with the real computed by φ^A. (So, for example, if we write "$A(n) = B(n)$," we mean "either $n \in A$ and $n \in B$, or $n \notin A$ and $n \notin B$." Likewise, if we write "$\varphi^A = B$", we mean "A computes B via the functional φ.") A real is also frequently thought of as an infinite sequence of zeroes and ones, and the nth bit of the real is the nth number in that sequence.

$2^{<\omega}$ is the set of finite binary strings. A binary string σ is a function $n \to 2$ for some n. σ can also be thought of as a sequence of zeroes and ones of length n. $|\sigma|$ is the domain of σ. $\#(\sigma)$ is the number of m such that $\sigma(m) = 1$, or in other words, the cardinality of σ, regarded as a subset of n. Such strings are frequently used as approximations to reals, and, if $|\sigma| = n$, we say that $\sigma \prec X$ if for every $m < n$, $\sigma(m) = X(m)$. In this case, we say that $\sigma = X \upharpoonright n$, and that σ is an *initial segment* of X. If $\sigma, \tau \in 2^{<\omega}$, and if $|\sigma| < |\tau|$, then we likewise say that $\sigma \prec \tau$, or that σ is an initial segment of τ if $\sigma(n) = \tau(n)$ for $n < |\sigma|$.

A *tree* is a set of finite sequences of natural numbers, closed under initial segment. A path through a tree is a real, thought of as an infinite sequence, each of whose initial segments is an element of the tree. Trees and other subsets of naturally countable sets will be referred to as "computable" if the preimages of those sets under the natural counting are computable. A real that is meant to be thought of as an actual real number will be referred to as a real number, or as an element of $\mathbb{R}$, or of $[0, 1]$.

Now, we begin with the definitions needed to discuss generic computability.

Definition 1.1.1. Let A be a real, and let $\alpha \in [0, 1]$. Then A has *density* α if the limit of the local densities of its initial segments is α, or in other words, if $\lim_{n\to\infty} \frac{\#(A\upharpoonright n)}{n} = \alpha$. In this case we will frequently say that A is *density-α*..

In this paper, we will mostly be interested in density-1 sets. Note that the intersection of two reals is density-1 if and only if each of the reals is density-1. (For any $\epsilon > 0$, once the local densities of each of the reals is $> 1 - \frac{\epsilon}{2}$, the density of their intersection will be $> 1 - \epsilon$.)

Sometimes, we will refer to the density of a real over an interval, I. In that case, it is understood that we mean $\frac{\#(A\upharpoonright I)}{\#I}$.

Definition 1.1.2. A real A is *generically computable* if there exists a partial recursive function φ whose domain has density 1, whose range is contained in $\{0,1\}$ such that if $\varphi(n)=1$ then $n \in A$, and if $\varphi(n)=0$ then $n \notin A$.

Notice that to generically compute a real, one is not allowed to make any mistakes. Density-1 many correct answers must be given, and no incorrect answers may be given. This is because the original motivation was not algorithms which sacrifice accuracy for speed, but rather algorithms which only give correct answers, and yet give answers much more quickly in general than in the worst case scenario. Thus, a generically computable set is a set which, almost everywhere, can be computed, and not a set which can be approximated very well by a computable set.

Definition 1.1.3. For reals A and B, A is *generically B-computable* if A is generically computable using B as an oracle. In this case, we frequently say B *generically computes* A, and we write $B \rightarrow_g A$.

Note here that we very intentionally do not use the notation "$B \geq_g A$." This is because the $\rightarrow_g$ relationship is very highly nontransitive (see Observation 1.1.9). In fact, in Proposition 1.2.5, we will see that $\rightarrow_g$ is as far from being transitive as possible. In Chapter 2, we will define and discuss *generic reduction*, a modified notion of relative generic computation which is transitive, and we reserve the symbol "$\geq_g$" for that notion.

Observations

To help dispel some common misconceptions, and to develop an intuition for generic computation, we prove a number of elementary facts concerning generic computation.

The following two definitions, and corresponding lemmas, will be very useful for these preliminary results.

Definition 1.1.4. Let A be a real. Then $\mathcal{R}(A)$ is the real given by $n \in \mathcal{R}(A) \leftrightarrow m \in A$, where 2^m is the largest power of 2 dividing n.

Definition 1.1.5. Let A be a real. Then $r(A)$ is the real given by $n \in r(A) \leftrightarrow \exists m(n = 2^m$ and $m \in A)$.

Lemma 1.1.6. *For any reals A and B, $B \geq_T A$ if and only if $B \rightarrow_g \mathcal{R}(A)$.*

Lemma 1.1.7. *For any real A, $r(A)$ is generically computable.*

Proof. The basic idea behind Definition 1.1.4 and Lemma 1.1.6 is that each bit of A is spread out across an entire "column" of $\mathcal{R}(A)$.

Each column has positive density, and so, in particular, any density-1 subset of $\mathbb{N}$ must intersect every column. Thus, to generically compute $\mathcal{R}(A)$, it is necessary to give at least one output in each column, and because *every* output of a generic computation must be correct, it is necessary to be able to correctly compute every bit of A.

Lemma 1.1.7 is trivially true, because for any A, $r(A)$ can be generically computed via the function φ where $\varphi(n) = 0$ if n is not a power of 2, and $\varphi(n)$ is undefined if n is a power of 2.

□

Thus, we immediately can conclude the following:

Observation 1.1.8. *Every nonzero Turing degree contains a real which is not generically computable, and a real which is generically computable.*

Proof. Let A be nonrecursive. Then $\mathcal{R}(A)$ is not generically computable, but $r(a)$ is generically computable.

It is clear that $A \equiv_T \mathcal{R}(A) \equiv_T r(A)$. □

This shows that although in some sense, generically computable reals are "closer" to being computable than other reals, this has nothing to do with the information content of the real, and everything to do with how the information is *distributed* within the real.

Also, we remind the reader that relative generic computation is not transitive, and in particular that it is important not to confuse the generic computational power of a real with the difficulty in generically computing the real.

Observation 1.1.9. *"A generically computes B" is not a transitive relationship.*

Proof. Let A be a nonrecursive real. Then $0 \rightarrow_g r(A)$ by Lemma 1.1.7, $r(A) \rightarrow_g \mathcal{R}(A)$ because $r(A) \geq_T \mathcal{R}(A)$, but $0 \nrightarrow_g \mathcal{R}(A)$ by Lemma 1.1.6 □

In fact, the generic computational power of a real is measured exactly by its usual computational power.

Observation 1.1.10. *$A \equiv_T B$ if and only if $\forall C(A \rightarrow_g C \iff B \rightarrow_g C)$.*

Proof. We prove the slightly stronger statement:

$A \geq_T B$ if and only if $\forall C(B \rightarrow_g C \implies A \rightarrow_g C)$.

Certainly, if $A \geq_T B$, then for any C, $A \rightarrow_g C$ if $B \rightarrow_g C$. This is by the usual proof of transitivity of Turing reductions: If $A \geq_T B$, and $B \rightarrow_g C$, then $A \rightarrow_g C$ by the algorithm which emulates the generic computation of C from B, while simultaneously computing B from A to determine the answers that the oracle for B would give.

Conversely, if $A \not\geq_T B$, then $A \nrightarrow_g \mathcal{R}(B)$ by Lemma 1.1.6. $B \rightarrow_g \mathcal{R}(B)$ because $B \geq_T \mathcal{R}(B)$. □

As our final observation of this section, we observe that the reason $r(A)$ is generically computable is not that $r(A)$ is density-0, but rather that $r(A)$ is a subset of a recursive density-0 set. In the following observation, we work with density-1 sets rather than density-0 sets, because they will be more useful to us in the future.

Observation 1.1.11. *Let A be a density-1 set. Then A is generically computable if and only if A contains a density-1 r.e. subset.*

Proof. If A contains a density-1 r.e. subset, B then A can be computed from the enumeration of B by the algorithm $\varphi(n) = \begin{cases} 1 & \text{if } n \in B \\ \text{undefined} & \text{otherwise} \end{cases}$. This algorithm halts on density-1 because B is density-1, and it never gives incorrect answers because $B \subseteq A$. Note that φ does not need to decide when it it going to be undefined: it simply does not halt on inputs that are not in B.

Conversely, if A is a density-1 set, and φ is a generic computation of A, then let $B = \{n \mid \varphi(n) = 1\}$. B is clearly r.e. It is contained in A because φ is not allowed to give incorrect answers, and it is density-1 because it is the intersection of A with $\text{dom}(\varphi)$, and the intersection of two density-1 sets is density-1. □

To generalize Observation 1.1.11, we first define a real to be *coarsely computable* if it agrees with a recursive real on a set of density-1.

Definition 1.1.12. A real A is *coarsely computable* if there exists a recursive real B such that $\{n \mid A(n) = B(n)\}$ is density-1.

Coarse computability is not immediately relevant to our study, but it will resurface in Chapters 2 and 3. Currently we mention it primarily to disambiguate between it and generic reduction.

Proposition 1.1.13. (Jockusch, Schupp) [6] *There exists a coarsely computable real that is not generically computable, and also a generically computable real that is not coarsely computable.*

We prove the first half here, and we defer the proof of the second half to Lemma A.0.17, in Appendix A, since our proof will use techniques that we introduce in Section 1.3, most notably, Definitions 1.3.5, and 1.3.6, and also Lemma 1.3.7.

The basic idea is that any density-1 real that has no density-1 r.e. subset is coarsely computable but not generically computable. Alternatively, if A is a Δ^0_2 real that is not recursive, then $\mathcal{R}(A)$ is coarsely computable but not generically computable.

Here, we present a construction of a density-1 real that has no density-1 subset.

We also stress that generic computability really is the computability notion that we wish to study - we are motivated by algorithms that usually give answers more quickly than they "should" be able to, not by algorithms that sacrifice accuracy in order to provide faster answers!

Proof. We build a real A such that A is density-1, and has no density-1 r.e. subset. In fact, it will not even have an infinite r.e. subset.

For each e, let W_e be the eth r.e. set. If W_e is infinite, let n_e be the smallest element of W_e that is greater than 2^e. If W_e is finite, let $n_e = 0$.

Then, let $A = \mathbb{N} \setminus \{n_e \,|\, e \in \mathbb{N}\}$.

Then first of all, A has no density-1 r.e. subsets, since A has no infinite r.e. subsets, since for every e, if W_e is infinite, then $n_e \in W_e$, and $n_e \notin A$.

Secondly, A is density-1, as follows.

For every m, A is missing at most $m+1$ of the elements less than 2^m. This means that if $2^m < n \leq 2^m + 1$, then

$$\frac{\#(A \restriction n)}{n} \geq \frac{n-(m+2)}{n} = 1 - \frac{m+2}{n} \geq 1 - \frac{m+2}{2^m}$$

which clearly approaches 1 as n (and thus implicitly m) goes to infinity.

□

1.2 Graph Embeddings

The primary purpose of this section will be to prove Theorem 1.2.5, which says that any reflexive binary relation with countable domain embeds into the reals under $\rightarrow_g$. This theorem shows that, in some sense, generic relative generic computation is as far from being transitive as possible, and indeed, its primary purpose is to help solidify the lesson from the observations in the previous section: that the set of reals that a given real can generically compute has nothing to do with the set of reals which can generically compute the given real.

Indeed, in the statement $A \rightarrow_g B$, the A and the B should not even truly be thought of as the same kind of thing. Lemma 1.1.10 shows that the A can be thought of in terms of its Turing degree without causing any confusion, but in Chapter 2, when we define generic reduction, we will see from Corollary 2.1.16, and Proposition 2.3.1 that B should be thought of in terms of its generic degree. (Observation 1.1.8 certainly shows that B should not be thought of in terms of its Turing degree!)

To prove Theorem 1.2.5, we will need two purely recursion theoretic facts.

Observation 1.2.1. *There exists a recursive reflexive binary relation R on $\mathbb{N}$ such that for any reflexive binary relation R' with countable domain, R' is isomorphic to R restricted to some subset of $\mathbb{N}$.*

The relation R is simply the Fraisse limit of the finite reflexive binary relations. We also present a direct construction of R:

Proof. We build our relation R in stages. At the end of any stage, R will be defined on a finite initial segment of $\mathbb{N}$.

At stage 0, the domain of R is $\{0\}$, and $\langle 0,0\rangle \in R$

At stage $s+1$, we extend R to be defined on the next 4^n many elements of $\mathbb{N}$, where n is the domain of R at the end of stage s. For every possible way to extend $R \restriction n$ by one element, we take one of the new elements and define R on that element via the chosen extension. Every new element is related to itself (to maintain reflexivitiy), and no new element is related to any other new element. (Note that since the relationship is not symmetric, we need 4^n many new elements. For every old element a, we must choose whether a new element b satisfies aRb and also independently whether it satisfies bRa.)

This construction can clearly be carried out recursively. Any countable reflexive binary relation R' embeds in it by picking a counting $\langle a_i\rangle$ of the domain of R', and mapping inductively each a_i to an element that was added to the domain of R at stage i. (By the construction of R, when the embedding has been defined on $a_0, ..., a_n$, there is exists a way to extend the embedding to map a_{n+1} to an element added at stage $n+1$.) □

Observation 1.2.2. *There exists a countable sequence of reals $\langle X_i\rangle$ such that for each i, X_i cannot be computed by the recursive join of the rest of the X_j.*

This is satisfied by the columns of any 1-random, or Cohen 1-generic subset of $\mathbb{N} \times \mathbb{N}$. This follows from the fact that no 1-random [3] or 1-generic is autoreducible. We present a proof of the existence of a nonautoreducible real for completeness.

Definition 1.2.3. A real A is autoreducible if there is a Turing functional φ such that for every n, $\varphi^A(n) = \begin{cases} 1 & \text{if } n \in A \\ 0 & \text{if } n \notin A \end{cases}$, and such that φ never queries the nth bit of A while computing the nth bit of A.

Lemma 1.2.4. *There exists a real A which is not autoreducible.*

Those familiar with forcing in arithmetic will recognize this proof as the proof that no Cohen 1-generic is autoreducible. For those familiar with randomness, a Kolmogorov-Loveland random is very clearly not autoreducible.

Proof. We build A by finite approximation.

At stage s, we diagonalize against A being autoreducible via φ_s in the following way.

Let $\sigma_s \in 2^<\omega$ be the approximation to A at stage s.

Choose n_s to be a number larger than $|\sigma_s|$.

Search for an extension τ of σ_s that will make $\varphi_s^\tau(n_s)$ halt without querying n_s.

If no such τ exists, then A cannot be autoreducible via φ_s.

Otherwise, let σ_{s+1} be equal to the τ that was found, except that $\sigma_{s+1}(n_s) = 1 - \varphi_s^\tau(n_s) = 0$. φ_s did not query n_s while computing n_s from τ, so $\varphi_s^\tau(n_s) = \varphi_s^{\sigma_{s+1}}(n_s)$, so $\sigma_{s+1}(n_s) \neq \varphi_s^{\sigma_{s+1}}(n_s)$, so as long as A is an extension of σ_{s+1}, A will not be autoreducible via φ_s. □

$n_s \in \sigma_{s+1} \leftrightarrow \varphi_s^{\sigma_{s+1}}(n_s) = 0$

With these two observations, we now prove our theorem.

Theorem 1.2.5. *For any reflexive binary relation R on $\mathbb{N}$, there exists a subset $\mathbf{X}$ of $\mathbb{R}$ such that $\langle \mathbb{N}, R\rangle$ is isomorphic to $\langle \mathbf{X}, \rightarrow_g\rangle$.*

The basic idea of this proof is that in each real, we code, with a large amount of redundancy, the information necessary to generically compute that real, and then we also code everything that it should be able to generically compute into a set of density-0.

Proof. By Observarion 1.2.1, it suffices to prove the proposition for recursive relations.

Fix a recursive reflexive relation R. Choose $\langle X_i\rangle$ as in Observation 1.2.2, and for each i, define Y_i to be the recursive join of the set of X_j such that $R(i,j)$. (More precisely $Y_i = \{\langle n, j\rangle | n \in X_j \wedge R(i,j)\}$.) Let $Z_i = \mathcal{R}(X_i) \oplus r(Y_i)$.

Let $\mathbf{X} = \{Z_i \mid i \in \mathbb{N}\}$.

If $R(i,j)$ then $Y_i \geq_T X_j$, so $Z_i \geq_T X_j$, so in particular we have that Z_i generically computes Z_j.

Conversely, if $\neg R(i,j)$ then Z_i is computable from the join of the X_k with $k \neq j$ ($\{j \mid R(i,j)\}$ is recursive, since R is a recursive relation). Thus in particular, $Z_i \not\geq_T X_j$, so $Z_i \not\twoheadrightarrow_g \mathcal{R}(X_j)$ by Lemma 1.1.6, so $Z_i \not\twoheadrightarrow Z_j$, because any density-1 set of bits of $\mathcal{R}(X_i) \oplus r(Y_i)$ must include density-1 many bits of $\mathcal{R}(X_i)$.

□

1.3 Minimal Pairs

At first glance, the lack of a degree structure for generic computation suggests that generic computation is not a realm in which it is fruitful to do recursion theory. However, as mentioned in the introduction to the previous section, the most major obstruction is that the oracle and the set being generically computed are, morally, different sorts of objects. If we regard them as such, then it becomes more clear which sorts of questions have natural analogues in this setting. For instance, we can ask whether there are minimal pairs for generic computation, by simply asking whether any two nonrecursive sets have something that is not generically computable that they can both generically compute. We could equivalently ask whether any two non-generically-computable sets have something they can both generically compute, but there is no reason to do this, given that the input sets should be thought of in terms of their Turing degrees. Likewise, in this question, the "output" is thought of only in terms of how difficult it is to generically compute without any reference to what the output set's computing power is. (In contrast, asking about a minimal degree would be awkward and probably unenlightening.)

The primary purpose of this section is to prove Theorem 1.3.13, which says that there are no minimal pairs for generic computation. (In fact, the proof can be modified to show that given any finite set of nonrecursive reals, there is a not-generically-computable real that every one of them can generically compute!) This result is somewhat startling, given that it implies that even if two sets form a minimal pair in the Turing degrees, they are still able to agree enough to both generically compute some non-generically-computable real. A key stepping stone will be our proof of Proposition 1.3.8, which says that given any nonrecursive set A, A can enumerate some density-1 set which has no density-1 r.e. subset. We will not need to use the actual proposition, but the techniques in the proof will be useful for the proof of Theorem 1.3.13.

Attempting to construct minimal pairs

In this subsection, we motivate Proposition 1.3.8 by explaining how its negation would imply the negation of Theorem 1.3.13. To do this, we start by describing the obstruction to adapting the usual minimal pair construction to the generic computation setting, and then we explain how a real not satisfying Proposition 1.3.8 would be able to overcome this obstruction, if it existed.

One of the primary purposes of this subsection is to help the reader understand the sorts of concerns that one must address when working with generic computation, and a reader looking for a proof of Theorem 1.3.13 may freely skip ahead to Lemma 1.3.4. However, the techniques of these proofs will be revisted in Chapter 2, when we address the existence of minimal pairs for generic reduction.

To familiarize the reader with the notation that will be used in this subsection, we first provide a brief proof of the existence of minimal pairs in the Turing degrees:

Theorem 1.3.1. *There exist nonrecursive reals A and B, such that for any real C, if $A \geq_T C$ and $B \geq_T C$, then C is recursive.*

Proof. We build A and B by finite approximation. We have one stage for each number e, and one for each pair of natural numbers, $\langle i, j\rangle$.

At each stage, we have a finite string $\sigma \in 2^{<\omega}$ that will eventually be an initial segment of A, and likewise, we have an approximation τ for B. We then extend these strings to longer finite strings, while accomplishing some task.

At stage e, we ensure that neither A nor B can be computed by φ_e.

To accomplish this, we choose an n longer than $|\sigma|$ or $|\tau|$. If $\varphi_e(n)$ does not halt, then we do not need to do anything, since φ_e is not a computation of anything. If $\varphi_e(n)$ halts, then extend σ and τ to strings $\tilde{\sigma}$ and $\tilde{\tau}$ such that $\tilde{\sigma}(n) = \tilde{\tau}(n) \neq \varphi_e(n)$. This guarantees that if A and B are extensions of $\tilde{\sigma}$ and $\tilde{\tau}$, then neither can be computed by φ_e.

At stage $\langle i, j\rangle$, we guarantee that if φ_i^A and φ_j^B both are computations of the same real, then that real is recursive.

First, if σ and τ can be extended to $\tilde{\sigma}$ and $\tilde{\tau}$ so that for some n, $\varphi_i^{\tilde{\sigma}}(n) \neq \varphi_j^{\tilde{\tau}}(n)$, then do that. This guarantees that when A and B are finished being built, $\varphi_i^A(n)$ will not be equal to $\varphi_j^B(n)$, and so, in particular, φ_i^A and φ_j^B will not both be computations of the same real.

If σ and τ cannot be extended to make φ_i and φ_j disagree anywhere, then we do not need to do anything, because if φ_i^A and φ_j^B both end up being computations of the same real, C, then we can compute C by the following method.

For any n, search for an extension $\tilde{\sigma}$ of σ such that $\varphi_i^{\tilde{\sigma}}(n)$ halts. We know we can find such a $\tilde{\sigma}$, because otherwise $\varphi_i^A(n)$ does not halt, and so φ_i^A is not a computation of anything.

Then n is in C if and only if $\varphi_i^{\tilde{\sigma}}(n) = 1$. The reason that we can conclude this is that we know B is an extension of τ, and we know σ and τ could not be extended to make φ_i and φ_j disagree, and so, because $\varphi_i^{\tilde{\sigma}}(n) = 1$, we know that if $\varphi_j^B(n)$ halts, then $\varphi_j^B(n) = 1$, and so $n \in C$.

Once all of the stages have been completed, we have ensured that neither A nor B is computable, and that if $A \geq_T C$, and $B \geq_T C$, then C is computable.

□

Now, we show why this proof does not work in the setting of relative generic computation.

Proposition 1.3.2. *The naive way to build a minimal pair for generic construction does not work.*

Of course, this proposition is difficult to make precise, and indeed, the most rigorous proof of the proposition will be the proof of Theorem 1.3.13. Here, our primary purpose is to isolate the flaw in the natural line of reasoning in order to clarify a major part of why generic computation does not work the way that Turing reduction works.

Proof. We attempt to adapt the proof of Theorem 1.3.1.

Again, we build A and B by finite approximation. Again, we have one stage for each e, and one for each $\langle i, j\rangle$.

We ensure that neither A nor B is computable the same way as before.

Then, at stage $\langle i, j\rangle$, we must extend σ and τ to make sure that if φ_i^A and φ_j^B both are generic computations of the same real, then that real is generically computable.

Again, if σ and τ can be extended to make φ_i and φ_j disagree, then we do that. Again, in this case, for the n at which they disagree, $\varphi_i^A(n)$ will not be equal to $\varphi_j^B(n)$, and so, in particular, there will not be any real that is generically computed both by φ_i^A and by φ_j^B.

Now, if σ and τ cannot be extended to make φ_i and φ_j disagree anywhere, and if, for some C, φ_i^A and φ_j^B are both generic computations of C, then we attempt to generically compute C as well.

Again, for any n, search for an extension $\tilde{\sigma}$ of σ such that $\varphi_i^{\tilde{\sigma}}(n)$ halts. We know that for density-1 many n's, we will find such a $\tilde{\sigma}$, because otherwise φ_i^A does not halt on density-1, and so φ_i^A is not a generic computation of anything. If we find such a $\tilde{\sigma}$, perhaps we try to be safe, and also find an extension $\tilde{\tau}$ of τ such that $\varphi_j^{\tilde{\tau}}(n)$ halts. Then, if we also find such a $\tilde{\tau}$, perhaps we decide to guess that n is in C if and only if $\varphi_i^{\tilde{\sigma}}(n) = \varphi_j^{\tilde{\tau}}(n) = 1$.

It seems safe, because at this point, we know that if $\varphi_i^A(n)$ halts, then it equals 1, and if if $\varphi_j^B(n)$ halts, then it equals 1. Unfortunately, it is entirely possible that neither $\varphi_i^A(n)$, nor $\varphi_j^B(n)$ halts, and that, indeed, $n \notin C$.

Our problem here is that our algorithm guarantees that our outputs are all *consistent* with the outputs of φ_i^A and φ_j^B, but, unfortunately, all that shows is that there *exists* a real C such that φ_i^A and φ_j^B are both generic computations of C, and such that we can generically compute C. It does not show that we can generically compute *every* C such that φ_i^A and φ_j^B are both generic computations of C.

□

This obstruction could be avoided by a counterexample to Proposition 1.3.8:

Proposition 1.3.3. *Let A be a nonrecursive real such that every density-1 set that A can enumerate has an r.e. subset which is density-1. Then there exists a nonrecursive real B such that for any real C, if $A \to_g C$ and $B \to_g C$, then C is generically computable.*

Proof. We build B by finite approximation as before.

We ensure that B is nonrecursive as before.

Given $\langle i, j\rangle$, and given an approximation τ for B, if there exists an extension $\tilde{\tau}$ of τ such that for some n, $\varphi_i^A(n) \neq \varphi_j^{\tilde{\tau}}(n)$, then we extend τ to $\tilde{\tau}$, thereby ensuring that φ_i^A and φ_j^B cannot both compute the same real.

If there is no such $\tilde{\tau}$, then, if φ_i^A and φ_j^B are both generic computations of C, then C is generically computable by the following algorithm, ψ.

$\{n \,|\, \varphi_i^A(n)\downarrow\}$ is r.e. in A, and it is density-1 because φ_i^A is a generic computation of C. So, by the assumption on A, fix a density-1 r.e. set $W \subseteq \{n \,|\, \varphi_i^A(n)\downarrow\}$.

Now, we define ψ so that $\psi(n)\downarrow$ if and only if $n \in W$ and there exists a $\tilde{\tau}$ extending τ such that $\varphi_j^{\tilde{\tau}}(n)\downarrow$. In this case, we let $\psi(n) = \varphi_j^{\tilde{\tau}}(n)$ for the first such $\tilde{\tau}$ that we find.

Then, because n is in W, we know that $\varphi_i^A(n)\downarrow$. Because τ could not be extended to make φ_i disagree with φ_j, we know that $\varphi_i^A(n) = \varphi_j^{\tilde{\tau}}(n) = \psi(n)$, so in particular, $n \in C \Leftrightarrow \psi(n) = 1$, because φ_i^A is a generic computation of C. Thus, ψ gives only correct answers on its domain. The domain of ψ is density-1 because it is the intersection of two density-1 sets. (The halting set of φ_i^A is density-1, and the set of n such that τ can be extended to make $\varphi_j^{\tilde{\tau}}(n)\downarrow$ is density-1 because it contains the halting set of φ_j^B.)

□

Of course, we will now see that there is no A satisfying the hypothesis of Proposition 1.3.3, and indeed that there are no minimal pairs (or even finite minimal sets) for generic computation.

Nonexistence of minimal pairs

Here, we prove Theorem 1.3.13, which says that that there are no minimal pairs for generic computation.

We saw, in the proofs of Propositions 1.3.2 and 1.3.3 that any proof of Theorem 1.3.13 will necessarily need to make careful use of the halting sets of the generic computations. Indeed, the halting sets of generic computations will be the only thing that is relevant to us, since all of our computations will output only 1s as answers, and this section will make liberal use of Observation 1.1.11, which we restate here as Lemma 1.3.4.

Lemma 1.3.4. *Let A be a density-1 set. Then A is generically computable if and only if A contains a density-1 r.e. subset.*

To prove Theorem 1.3.13, given two nonrecursive reals A and B, we will show that A and B can each enumerate a density-1 set such that the union of those two density-1 sets has no density-1 r.e. subset. We will do this in three parts, Propositions 1.3.9, 1.3.11, and 1.3.12, depending on whether neither, one or both of the two sets are Δ^0_2. We will require a technical lemma to adapt our proof of Proposition 1.3.9 to prove Propositions 1.3.11 and 1.3.12. Proposition 1.3.12 has already been proved by Downey, Jockusch, and Schupp [2], but the technical lemma that we use to prove Proposition 1.3.11 proves Proposition 1.3.12 as well.

We begin by introducing some terminology that will be used for the proofs.

Definition 1.3.5. *Let $P_i = \{n \in \mathbb{N} \mid 2^i \leq n < 2^{i+1}\}$.*

Note then that $\mathbb{N}$ is the disjoint union of the P_i together with $\{0\}$.

Definition 1.3.6. *For $X \subseteq \mathbb{N}$, we say that X has a gap of size 2^{-e} at P_i if the last 2^{i-e} many elements of P_i are not elements of X.*

Note then the following lemma:

Lemma 1.3.7. *If the only elements missing from X are from gaps of the form just described, then X is density-1 if and only if for every e, X has only finitely many gaps of size 2^{-e}*

Proof. If X has a gap of size 2^{-e} at P_i then $\frac{\#(X\restriction 2^i)}{2^i} \leq 1 - 2^{-e-1}$, so in particular, if X has infinitely many gaps of size 2^{-e}, it does not have density 1.

Conversely, if there is some i such that after P_i, all of the gaps in X have size $\leq 2^{-e}$, then for $n \geq 2^{i+1}$, the density of X over the interval $[2^i, n]$ will be $\geq 1 - 2^{-e}$, and so the density of $X \restriction n$ will always be $\geq 1 - 2^{-e+1}$. (Note that since the gaps appear at the ends of the P_i, the local minima of the density of X always occur at the end of a P_i.) If for every e, there exists such an i, then the limiting density of X will be 1. $\square$

Now, we prove Proposition 1.3.8. This proof will be generalized to allow us to prove Propositions 1.3.9, 1.3.11, and 1.3.12, but we will only go through it in full detail here.

Proposition 1.3.8. *For any nonrecursive real A, there is a density-1 set $S(A)$ that is r.e. in A such that $S(A)$ has no density-1 r.e. subset. In fact, $S(A)$ can be found to be recursive in A, and even tt-reducible to A.*

First, a brief overview: we will define a total Turing functional φ on 2^ω. For each e, there will be a strategy that diagonalizes against the eth r.e. set, W_e, being a density-1 subset of φ^X for any real X. In doing so, the eth strategy will cause at most one real X_e to have the property that φ^{X_e} is not density-1. X_e will be the leftmost path through a recursive tree T_e, so repeating the same argument with rightmost paths gives a pair of functionals φ and ψ such that for any nonrecursive X either φ^X or ψ^X is density-1.

Proof. Over the course of the construction, after stage s, $\varphi^X \upharpoonright 2^s$ will be defined for every X, using at most the first s bits of X. This will be useful in verifying that the strategies work as they are supposed to.

The eth strategy acts as follows:

Define T_e as the tree whose infinite paths consist of the reals X such that $W_e \subseteq \varphi^X$. Note that T_e is a recursive tree, since we can determine the lth level of T_e in the following way. Run the enumeration of W_e for the first l steps. For every n less than 2^l, if W_e has enumerated n, remove any σ of length l such that $\varphi^\sigma(n) = 0$. As long as l is less than the stage s of the construction, this can be accomplished recursively. Let $T_{e,s}$ be the tree consisting of all extensions of the $(s-1)$th level of T_e.

At stage s, if $s < e$, do nothing. Also, if $T_{e,s}$ is finite, do nothing. Else, place a marker p_s on the leftmost infinite path of $T_{e,s}$, at the shortest σ on that path such that σ has no marker. Then define $\varphi^X \upharpoonright P_s$ for all X by placing a gap of size 2^{-e} into φ^X at P_s if $\sigma \prec X$, and by not placing such a gap if $\sigma \not\prec X$. (Define φ so that φ^X has a gap of size 2^{-e} at P_s if and only if $\sigma \prec X$.)

The idea here is that the marker p_s signifies the existence of a trap at P_s: W_e must either avoid enumerating any of the elements of the gap at P_s, thereby creating another instance of its density dropping below $1 - 2^{e+1}$, or it must enumerate some of those elements, thereby removing all extensions of σ from the tree T_e.

Note then that if T_e is finite, then for every X, φ^X has only finitely many gaps of size 2^{-e}. Furthermore, in this case, the strategy has guaranteed that $W_e \nsubseteq \varphi^X$ for any X. (After the stage at which the tree is seen to be finite, the eth strategy stops acting. This stage is precisely the point at which, for every X, W_e has enumerated something not in φ^X.)

On the other hand, if T_e is infinite, then the leftmost path, X_e, of T_e has infinitely many markers on it, so φ^{X_e} has infinitely many gaps of size 2^{-e} so in particular, W_e is not density-1. ($W_e \subset \varphi^X$ for every infinite path X through T_e. This is because of how T_e is defined.) Furthermore, if $X \neq X_e$ then X has only finitely many markers on it, so φ^X has only finitely many gaps of size 2^{-e} (If $X \neq X_e$ then there is some stage s and some $\sigma \prec X$ such that after stage s, σ never looks like it might be an initial segment of the leftmost path of T_e, so after that stage s, X can only get at most $|\sigma|$ many markers placed on it)

Finally, note that the strategies have no need to interact: they ignore each other's markers and trees, and at stage s, at most $s+1$ many strategies are eligible to act, and φ^X gets defined on P_s for every X by just defining φ^X to be the intersection of the sets that each strategy wants φ^X to be. By Lemma 1.3.7, a real X will have the property that φ^X is not density-1 if and only if some specific strategy causes φ^X to not be density-1. Thus, the only reals X such that φ^X is not density-1 are the X_e.

As mentioned in the overview, repeating the same construction again with rightmost paths will finish the proof, since if X is the leftmost path of one recursive tree and the rightmost path of another, then X is recursive. If this is not the case, then for one of the two constructions, the set computed from X is density-1 and has no density-1 r.e. subset.

$\square$

Next, we proceed to prove Proposition 1.3.9. The proof is effectively the same as the proof of Proposition 1.3.8, with the only major modification being that we define two functionals simultaneously, and replace T_e with a 4-ary branching tree whose paths correspond to pairs of reals $\langle X, Y\rangle$ such that $W_e \subset \varphi^X \cup \psi^Y$.

Proposition 1.3.9. *If A and B form a minimal pair for generic computability, then either A or B is Δ^0_2.*

Proof. Again, we describe how the eth strategy acts at stage s:

If $s < e$, do nothing. Otherwise define $T_{e,s}$ as the set of pairs $\langle\sigma, \tau\rangle$ such that $|\sigma| = |\tau|$ and such that $W_{e,s} \upharpoonright 2^{s-1} \subset \varphi^X_{s-1} \cup \psi^Y_{s-1}$ for some $X \succ \sigma$, and some $Y \succ \tau$. Note that this will be recursive, since we define φ and ψ restricted to P_s by the end of stage s.

Then, if $T_{e,s}$ is finite, do nothing. Otherwise put a marker p_s on the leftmost infinite path of $T_{e,s}$, at the shortest pair $\langle\sigma, \tau\rangle$ on that path such that $\langle\sigma, \tau\rangle$ has no marker. Then define $\varphi^X_s \upharpoonright P_s$ and $\psi^Y_s \upharpoonright P_s$ for all X and Y by placing a gap of size 2^{-e} into φ^X_s at P_s if $\sigma \prec X$, and by not placing such a gap if $\sigma \not\prec X$, and likewise placing a gap of size 2^{-e} into ψ^Y_s at P_s if and only if $\tau \prec Y$.

Note, as before, that only one path through T_e gets infinitely many markers on it, so in particular only finitely many markers are placed on nodes that are not on that path. That path corresponds to a pair of reals $\langle X_e, Y_e\rangle$, and if $X \neq X_e$ then φ^X will have only finitely many gaps of size 2^{-e}. Note also that the leftmost path of T_e computes both X_e and Y_e, so in particular, both are Δ^0_2, but it is not true that either is necessarily the leftmost path of a recursive tree, so we are unable to use the previous trick to get them to be recursive.

Again, the strategies do not interfere with each other, and so if A is different from all of the X_e and B is different from all of the Y_e then φ^A is density-1, ψ^B is density-1, and $\varphi^A \cup \psi^B$ has no density-1 r.e. subset. By Lemma 1.3.4, this suffices, since $\varphi^A \cup \psi^B$ is not generically computable, but A and B can each generically compute it, because they can each enumerate (and in fact, compute) a density-1 subset of $\varphi^A \cup \psi^B$. A is different from all of the X_e and B is different from all of the Y_e, because neither A nor B is Δ^0_2, while all of the X_e and Y_e are Δ^0_2.

□

Now we prove our technical lemma, which states that the "leftmost path" in the above construction can be replaced by any uniformly chosen Δ^0_2 infinite path through T_e.

Lemma 1.3.10. *Let $\mathcal{F}$ be any function from reals to reals such that for a 4-ary branching recursive tree T, if T is infinite, then $\mathcal{F}(T)$ is an infinite path through T, and such that $\mathcal{F}(T)$ is uniformly recursive in $0'$ together with a recursive index for T. Then for any reals A and B, one of the following three things holds.*

1: A and B do not form a minimal pair for generic computability.

2: There exists a recursive tree T such that $\mathcal{F}(T) \geq_T A$.

3: There exists a recursive tree T such that $\mathcal{F}(T) \geq_T B$.

So, for example, letting $\mathcal{F}$ be the function corresponding to the construction in the proof of the low basis theorem, we could prove that if A and B form a minimal pair for generic computability, then either A or B must be low.

Proof. We modify the proof of Proposition 1.3.9 by, for each e, choosing a Δ^0_2 index for $\mathcal{F}(T_e)$. This can be done uniformly by the fixed point theorem, since we can uniformly compute the trees T_e from the graphs of φ and ψ, and since $\mathcal{F}$ is assumed to be uniform. (By the fixed point theorem, we may assume we have a fixed index for the graphs of the Turing functionals φ, ψ that we build. The graphs are recursive, as, at stage s, $\varphi^X(n), \psi^Y(m)$ are defined for all X and Y and for all $n, m \leq 2^s$. T_e is uniformly computable from the graphs of φ and ψ, and a Δ^0_2 index for $\mathcal{F}(T_e)$ is uniformly computable from T_e by assumption.)

Having chosen a Δ^0_2 index for each $\mathcal{F}(T_e)$, at each stage, instead of placing a marker on the shortest unmarked node of the leftmost infinite path of T_e, the eth strategy places a marker on the shortest unmarked node of the current approximation to $\mathcal{F}(T_e)$. We then proceed to place the corresponding gaps in φ and ψ in the usual way. The eth strategy does nothing if $T_{e,s}$ is finite.

Then, as before, if T_e is infinite, then the only path that gets infinitely many markers is $\mathcal{F}(T_e)$. This is because for any n, after the first n bits of $\mathcal{F}(T_e)$ have stabilized to their final configuration, all future markers will be on extensions of $\mathcal{F}(T_e) \upharpoonright n$. In this case, W_e is not density-1, so the eth strategy succeeds. If T_e is finite, then for any X and for any Y, $W_e \nsubseteq \varphi^X \cup \psi^Y$ as before.

Then, for each e, define X_e and Y_e as before, note that $\mathcal{F}(T_e)$ computes either of them, and note that φ^X and ψ^Y both compute density-1 subsets of the same non-generically computable real, as long as for every e, $X \neq X_e$ and $Y \neq Y_e$.

□

We now can prove Proposition 1.3.11 as a direct corollary of this lemma.

Proposition 1.3.11. *If A is Δ^0_2 and B is not Δ^0_2 then A and B do not form a minimal pair for generic computability.*

Proof. For any infinite recursive tree T, and any nonrecursive Δ^0_2 set A, $0'$ can uniformly (in indices for T and A) compute an infinite path Z through T such that $Z \ngeq_T A$. (See Theorem B.0.18 in Appendix B for a proof of this fact.)

Apply Lemma 1.3.10 using the $\mathcal{F}$ representing this computation, and note then that for any T, $\mathcal{F}(T) \ngeq A$ by construction. Also, $\mathcal{F}(T) \ngeq B$ because B is not Δ^0_2, and $\mathcal{F}(T)$ is. Therefore, A and B do not form a minimal pair for generic computability.

□

Note that this same proof can be modified to prove Proposition 1.3.12, by simultaneously avoiding the cones above both A and B.

Proposition 1.3.12. (Downey, Jockusch, and Schupp) [2] *If A and B are both Δ^0_2 then A and B do not form a minimal pair for generic computability.*

Now, we combine Propositions 1.3.9, 1.3.11, and 1.3.12 to prove our main theorem for the section.

Theorem 1.3.13. *For any nonrecursive reals A and B, there exists a real C such that C is not generically computable, but such that C is both generically A-computable and generically B-computable. Thus, there are no minimal pairs for generic computation.*

In fact, we can easily strengthen our construction to accommodate larger finite sets of reals, and we can show that for any n, there are no minimal n-tuples for generic computation.

Theorem 1.3.14. *For any nonrecursive reals $A_0, ..., A_{n-1}$, there exists a real C such that C is not generically computable, but such that for every $i < n$, $A_i \rightarrow_g C$.*

Proof. We use the same construction as for Propositions 1.3.9, 1.3.11, and 1.3.12, except that we use a 2^n-ary branching tree, n different functionals, and we simultaneously avoid the cones above all of the Δ^0_2 sets among the $A_0, ..., A_{n-1}$. The details are omitted.

□

Chapter 2

Generic Reduction

2.1 Introduction

We have seen that relative generic computability is a wildly nontransitive relation, and we have commented that the reason for this is that in a relativized generic computation, the input reals and the output reals are not really the same sort of thing. Here, we seek to deepen our understanding of generic computation by generalizing it to a transitive notion of computation, generic reducibility. The most important property that one might expect of this generalization would be to preserve the definition of the generically computable sets, and in particular a generic reduction using a generically computable oracle should be the same as a generic computation.

We hope that by analyzing generic reducibility, we will acquire a more thorough understanding of generic computability, both because it will help us understand what can be done if one knows a generic description of a real (and thus what can be gained from generically computing a real), and because, once we have a degree structure, we have a larger array of tools at our disposal for our study. As with generic computation, we also engage in this study because it is surprisingly strange, counterintuitive, and foreign, and we hope to widen our understanding of computation in general by understanding computation in this context.

In this section, we introduce four notions of generic reducibility, prove that two of them are equivalent, and prove that at least two of them are distinct. We also prove a few general results that hold for all our notions of generic reducibility.

We caution the reader that our notation here differs slightly from the notation of our previous paper [5] on generic computation in order to make our notation agree better with the notation used by others in the field: Jockusch and Schupp [6] use $\leq_g$ for generic reduction. We previously used $\leq_G$ for this notion, but now we will use $\leq_g$, and we will reserve $\leq_G$ for nonuniform generic reduction.

Definitions

Definition 2.1.1. A *generic description* of a real A is a set S of ordered pairs $\langle n, x\rangle$, with $n \in \mathbb{N}, x \in \{0,1\}$, such that:
if $\langle n, 0\rangle \in S$ then $n \notin A$,
if $\langle n, 1\rangle \in S$ then $n \in A$,
and $\{n \mid \exists x\langle n, x\rangle \in S\}$ is density-1.

Thus, in particular, the graph of a generic computation is a generic description. It should be mentioned that this notation conflicts slightly with the notation of Jockusch and Schupp [6], in that they define a generic description as a partial function, so that, by their definition, a generic computation is simply a generic description which is partial recursive. We define "generic description" as a set, since the terminology of recursion theory is well suited to discussing sets are inputs and outputs of reductions. However, in this paper, sets are frequently referred to as if they were functions, so if, for example, we refer to the domain of a generic description, or the outputs of a generic description, we assume that the meaning will be clear from context. Sometimes, when a generic description is used as an oracle, we will refer to it as a "generic oracle." A generic oracle for A will frequently be denoted as (A).

It is interesting to notice, though, that the output of a generic computation of A is more than just a generic description of A, in that it is an *enumeration* of a generic description of A. For this reason, we define:

Definition 2.1.2. A *time-dependent generic description* of a real A is a set S of ordered triples $\langle n, x, l\rangle$, with $n, l \in \mathbb{N}, x \in \{0,1\}$, such that $\{\langle n, x\rangle \mid \exists l\langle n, x, l\rangle \in S\}$ is a generic description of A.

So a time-dependent generic description of A is an enumeration of a generic description of A, together with a record of when the elements of that generic description were enumerated. Thus, B generically computes A if and only if B can enumerate a generic description of A, which is true if and only if B can compute a time-dependent generic description of A. Here, l should be thought of as the stage at which it is known whether or not $n \in S$. It is frequently convenient to assume that for each n, x there is at most one l in the description. Adding this assumption presents no complications.

Jockusch and Schupp [6] define generic reducibility via enumeration operators. The relation is basically the one that one might intuitively construct, using only densely much information from the oracle, A, to deduce a generic computation of B. For those familiar with the notation, the definition is as follows:

Definition 2.1.3. A *generic reduction* of B from A is an enumeration operator which, given any generic description of A as input, outputs a generic description of B. B is *generically reducible to* A if there exists a generic reduction of B from A. In this case, we write $A \geq_g B$.

For those unfamiliar with the notation,

Definition 2.1.4. An *enumeration operator* is an r.e. set W of codes for pairs $\langle n, D\rangle$ where $n \in \mathbb{N}$ and D is a code for a finite subset of $\mathbb{N}$. It is thought of as a function, sending a real A to the set $\{n \mid \exists D[D \subseteq A \wedge \langle n, D\rangle \in W]\}$.

It should be noted that this definition of generic reduction has the following features. The computation of B from A must be uniform in the generic description of A, and the computation is only allowed to reference which sets are contained in the graph of the input set when computing a generic description for the output set (so in particular, giving less information about A never results in more information about B, and information is not allowed to be deduced from the rate/order of enumeration of the graph of A).

With these comments in mind, we make the following definitions:

Definition 2.1.5. A *time-dependent generic reduction* of B from A is a Turing functional which, given any time-dependent generic description of A as input, outputs a time-dependent generic description of B. B is *time-dependently generically reducible to* A if there exists a time-dependent generic reduction of B from A.

Definition 2.1.6. B is *non-uniformly generically reducible to* A if for every generic description of A, there is an enumeration operator which outputs a generic description of B using the given generic description of A as input.

Definition 2.1.7. B is *non-uniformly time-dependently generically reducible to* A if every time-dependent generic description of A, can compute a time-dependent generic description of B. In this case, we write $A \geq_G B$.

Again, we mention that the ability to compute a time-dependent generic description of a set is equivalent to the ability to enumerate a generic description of the set, and so the difference between the time-dependent and non-time-dependent reductions is entirely a difference in terms of what the input of the reduction is. The outputs are phrased differently just to make transitivity obvious, and also to make it easier to work with any given form of reduction on its own. Also for the remainder of this paper, whenever a non-time-dependent generic description is used as an oracle, it will be assumed that it is being used in the context of a generic reduction, and in particular, if neither $\langle n, 0\rangle$, nor $\langle n, 1\rangle$ is in the generic description, we will not be allowed to "know" that fact.

Equivalences and nonequivalences of definitions

From the definitions, we may immediately conclude the following implications.

Observation 2.1.8. *The existence of a uniform reduction of either type implies the existence of a non-uniform reduction of the corresponding type.*

Observation 2.1.9. *The existence of a non-time-dependent reduction of either type implies the existence of a time-dependent reduction of either type.*

Observation 2.1.8 is trivially true. Observation 2.1.9 is true since from any time-dependent generic description, one can enumerate a generic description, and then proceed to use the corresponding non-time-dependent reduction procedure. Thus, if there exists a non-time-dependent reduction, that reduction also functions as a time dependent one. The rules of enumeration operators do not allow for an obvious converse to Observation 2.1.9, although the first proposition that we prove is that the converse does hold for the uniform generic reductions, and so in particular, the two uniform reductions are equivalent.

Proposition 2.1.10. *$A \geq_g B$ if and only if the following holds:*

There is a Turing functional φ such that for any time-dependent generic description X of A, φ^X is a generic computation of B.

By Observation 2.1.9, we need only prove that the second implies the first.

Proof. Assume that from every time-dependent generic description X of A, φ^X is a generic computation of B. Then in particular, there are no time-dependent generic descriptions X, of A, such that φ^X gives false information about B. So, given a generic oracle (A) for A, we may generically compute B from (A) by considering all time-dependent versions X of (A), and outputting anything that φ^X would output on any of those inputs.

More formally, let W be the set of all $\langle a, D\rangle$ such that a codes an ordered pair $\langle x, y\rangle$, D is a finite set of ordered pairs $\langle n_i, m_i\rangle, i \leq c$, and such that there exists a sequence $\langle l_i \mid i \leq c\rangle$ where φ^σ gives output y on input x for $\sigma = \{\langle\langle n_i, m_i\rangle, l_i\rangle \mid i \leq c\}$.

Then, for any generic description of A, the output of W will be the set of all pairs $\langle x, y\rangle$ such that $\varphi^X(x) = y$ for some time-dependent version, X, of that generic description. The domain of this partial function will be density-1, because it is the union of the domains of the φ^X, every one of which is density-1. The outputs of this partial function will all be correct, because φ^X never gives incorrect answers. Thus, the output of W will be a generic description of B. $\square$

Next, we show that neither of the nonuniform reductions is equivalent to the uniform reduction. To prove this, we introduce some notation. Recall from Chapter 1, the definition of $\mathcal{R}(X)$:

Definition 2.1.11. For any real X, $n \in \mathcal{R}(X) \leftrightarrow m \in X$, where 2^m is the largest power of 2 dividing n.

Note now the following strengthening of Lemma 1.1.6, proved by Jockusch and Schupp [6], but with a proof included for completeness:

Observation 2.1.12. *For any real X, X computes $\mathcal{R}(X)$ uniformly and X can be computed uniformly from any generic description of $\mathcal{R}(X)$. Therefore, the map sending $X \mapsto \mathcal{R}(X)$ induces an embedding from the Turing degrees to the generic degrees.*

Here, the generic degrees are the equivalence classes of the reals under mutual generic reducibility, together with the partial ordering induced by generic reduction. These degrees will depend on the definition of generic reduction that is used, but the proof is sufficiently general to work for all four of the generic degree structures.

Proof. X computes $\mathcal{R}(X)$ uniformly, and so generically computes $\mathcal{R}(X)$ uniformly.

Conversely, to compute the mth bit of X from a generic description of $\mathcal{R}(X)$, search for any n such that 2^m is the largest power of 2 dividing n and such that the generic description of $\mathcal{R}(X)$ has a value for the nth bit of $\mathcal{R}(X)$. Use that as the value for the mth bit of X. There must be such an n because the set of numbers divisible by 2^m and not by 2^{m+1} has positive density (in fact, has density $\frac{1}{2^{m+1}}$), and the generic description has values for density-1 many bits of $\mathcal{R}(X)$.

The proof of the embedding follows directly: If X computes Y then any generic description of $\mathcal{R}(X)$ can be used uniformly to compute X and therefore Y and therefore $\mathcal{R}(Y)$. Likewise, if $\mathcal{R}(X) \geq_g \mathcal{R}(Y)$, then X can compute $\mathcal{R}(X)$, which can generically compute $\mathcal{R}(Y)$. Y can then be recovered from this generic description. □

We now introduce an alternate definition, which would have sufficed for the purposes of Lemma 1.1.6, but with the property that one cannot uniformly recover X from a generic description of $\widetilde{\mathcal{R}}(X)$.

Definition 2.1.13. For any real X, $n \in \widetilde{\mathcal{R}}(X) \leftrightarrow m \in X$, where 2^m is the largest power of 2 less than or equal to n. (0 is never in $\widetilde{\mathcal{R}}(X)$).

The key distinction here is that $\mathcal{R}(X)$ codes each of the entries of X into a positive density set, and so any generic description of $\mathcal{R}(X)$ must be able to recover all the entries of X. On the other hand, $\widetilde{\mathcal{R}}(X)$ codes the entries of X into progressively larger sets, each finite, and so any generic description of $\widetilde{\mathcal{R}}(X)$ is only guaranteed to be able to recover all but finitely many of the entries of X.

Proposition 2.1.14. *Let A be a real such that one cannot uniformly compute A from an arbitrary cofinite subset of the entries of A. Then $\mathcal{R}(A)$ is non-uniformly generically equivalent to $\widetilde{\mathcal{R}}(A)$ (and therefore non-uniformly time-dependently generically equivalent), but $\widetilde{\mathcal{R}}(A) \not\geq_g \mathcal{R}(A)$. Therefore, neither of the non-uniform notions of generic reducibility is equivalent to the uniform notion.*

Note that there exist reals satisfying the hypothesis of the proposition, for example any 1-random real, or any Cohen 1-generic real. Alternatively, Lemma 1.2.4 provides us with a real that is not autoreducible, and any real that is not autoreducible will satisfy this hypothesis.

Proof. First of all, any generic description of $\mathcal{R}(A)$ can be used uniformly to recover A by Observation 2.1.12. It can therefore compute $\widetilde{\mathcal{R}}(A)$.

Likewise, from any generic description of $\widetilde{\mathcal{R}}(A)$, one can uniformly compute all but finitely many bits of A. From this, one can non-uniformly compute A (by directly coding the missing bits into the computation), and therefore compute $\mathcal{R}(A)$.

On the other hand, any cofinite subset of the entries of A can be used to uniformly compute a generic description of $\widetilde{\mathcal{R}}(A)$. Any generic description of $\mathcal{R}(A)$ can be used to uniformly compute A. Thus, since there is no uniform way to compute A from a cofinite subset of its entries, there is no uniform way to go from a generic description of $\widetilde{\mathcal{R}}(A)$ to a generic description of $\mathcal{R}(A)$. □

We note now one important corollary of Observation 2.1.12

Corollary 2.1.15. *For any reals A, and B, A generically computes B if and only if B generically reduces to $\mathcal{R}(A)$.*

Proof. This follows directly from the statement of Observation 2.1.12. □

This allows us to conclude the comment at the beginning of Chapter 1, Section 2, that the output of a generic computation should be thought of in terms of its generic degree:

Corollary 2.1.16. *If A and B are generically equivalent, then for any C, $C \to_g A$ if and only if $C \to_g B$.*

Proof. If A and B are in the same generic degree, then they also have the same generic degrees above them. Thus $\mathcal{R}(C) \geq A$ if and only if $\mathcal{R}(C) \geq B$. Here, $\geq$ stands for any of the notions of generic reduction.

□

Finally, we mention and prove a basic lemma that will be important to us.

Lemma 2.1.17. *Let A and B be reals. Then the generic degree of $A \oplus B$ is the least upper bound of the generic degrees of A and B.*

The proof of this lemma is basically that a generic description of $A \oplus B$ is the same as a generic description of A together with a generic description of B. The proof is simply the process chasing inequalities to verify that fact.

Proof. To generically compute A from a generic oracle (C) for $A \oplus B$, whenever (C) gives an output on an even number $2n$, $\varphi^{(C)}(n)$ halts and gives that same output. Then, certainly φ never gives any false outputs, so we only need to verify that if $\mathrm{dom}((C))$ is density-1, then $\mathrm{dom}(\varphi^{(C)})$ is density-1.

So, assume $\mathrm{dom}((C))$ is density-1.

Let $\epsilon > 0$. Fix m such that for all $n \geq m$, $\frac{\#(\mathrm{dom}((C))\restriction n)}{n} > 1 - \frac{\epsilon}{2}$.

Then, since the number of odd bits in $\mathrm{dom}((C)) \restriction n$ can be at most $\frac{n}{2}$, we have that, if $f(n)$ is the number of even bits in $\mathrm{dom}((C)) \restriction n$, then $\frac{f(n)}{n} > \frac{1}{2} - \frac{\epsilon}{2}$. Thus $\frac{f(n)}{n/2} > 1 - \epsilon$. So, for all $k \geq \frac{m}{2}$, $\frac{\#(\mathrm{dom}(\varphi^{(C)})\restriction k)}{k} = \frac{f(2k)}{k} > 1 - \epsilon$.

Similarly, the natural computation of the odd bits also halts on density-1.

Now, to show that if $D \geq_g A$, and $D \geq_g B$, then $D \geq_g A \oplus B$ (here, we are using $\geq_g$ as shorthand for any of our notions of generic reduction), first we mention that any generic oracle, (D), for D can be used to generically compute $A \oplus B$ by simply simultaneously generically computing A and generically computing B, and outputting the bits of A on the even bits, and the bits of B on the odd bits.

This clearly only gives correct outputs about $A \oplus B$, so it remains to check that the domain of this computation is density-1.

For this, assume that (D) generically computes A via φ_0, and generically computes B via φ_1. Let ψ be the natural reduction which combines the two reductions into a reduction for $A \oplus B$.

Let $\epsilon > 0$.

Fix m such that for all $n \geq m$, $\frac{\#(\mathrm{dom}(\varphi_0^{(D)})\restriction n)}{n} > 1-\epsilon$, and $\frac{\#(\mathrm{dom}(\varphi_1^{(D)})\restriction n)}{n} > 1-\epsilon$.

Then, we claim that for $n \geq 2m$, $\frac{\#(\mathrm{dom}(\psi^{(D)})\restriction n)}{n} > 1-\epsilon$.

This is simply because

$$\#\left(\mathrm{dom}\left(\psi^{(D)}\right)\restriction n\right) = \#\left(\mathrm{dom}\left(\varphi_0^{(D)}\right)\restriction \left\lceil \frac{n}{2} \right\rceil\right) + \#\left(\mathrm{dom}\left(\varphi_1^{(D)}\right)\restriction \left\lfloor \frac{n}{2} \right\rfloor\right),$$

and the two summands, when divided by $\left\lceil \frac{n}{2} \right\rceil$, and $\left\lfloor \frac{n}{2} \right\rfloor$, respectively are each greater than $1-\epsilon$.

(This uses the fact that, for $a, b, c, d > 0$, if $\frac{a}{b} > x$ and $\frac{c}{d} > x$, then $\frac{a+c}{b+d} > x$.)

□

Notations for working with partial oracles.

Having established the formalisms and basics concerning the definitions of our reductions, we mention some more colloquial terminology that will be useful to us, and prove some basic lemmas concerning that terminology.

Definition 2.1.18. A *partial oracle* (A) for A is an oracle which, when asked whether or not $n \in A$, is not obligated to always answer, is not obligated to answer immediately if it will answer, and is not obligated to say whether or not it will give an answer. However, it is obligated to only give correct answers when it does give answers. The *domain* of the partial oracle is the set of n such that (A) gives an answer when asked whether or not $n \in A$. If n is in the domain of (A), we say that (A) *halts on* n.

When using a partial oracle in a uniform fashion, or in a nonuniform time-independent fashion, a Turing functional may query a bit of the oracle, and then keep doing other work while it waits to see if the oracle will respond. However, everything that the functional outputs must be able to be phrased in the form "If (A) shows that the following facts are true about A, then output the following...". (The functional is not allowed to use clauses of the form "If (A) has not yet halted on this input..." or "If (A) halts on this input before it halts on that input...".) Note that this is simply an informal way of thinking of enumeration reductions.

If there are contradictory facts that are concluded by a Turing functional, for instance, if $\varphi^{(A)}(5) = 0$, and $\varphi^{(A)}(5) = 1$ can both be concluded from (A), then we simply say that φ is *ill-defined* on (A) at 5.

Note that, in general, it will not present a problem that Turing functionals can be ill-defined if they are equipped to be able to work with partial oracles. This is primarily because, when defining a generic reducibility $A \geq_g B$, we always need to prove that the reduction never makes any mistakes on any partial oracle for A, and, in particular, this shows that the reduction is not ill-defined on any oracle for A, since if it gives any output, that output must be correct. Likewise, if proving that $A \not\geq_g B$, if we find a partial oracle (A) such that $\varphi^{(A)}(n) \neq B(n)$, then we already know that φ is not a generic reduction of B from A, regardless of whether there are also partial oracles for A which give correct answers.

Note also that the rules of partial oracles imply that if $(A)_1$ and $(A)_2$ are partial oracles for A, and if dom$((A)_1)$ $\subseteq$dom$((A)_2)$, then for every n, if $\varphi^{(A)_1}(n) \downarrow= k$, then $\varphi^{(A)_2}(n) \downarrow= k$. (However, it is entirely possible that $\varphi^{(A)_1}$ is well defined at n, but $\varphi^{(A)_2}$ is ill-defined at n. Again, though, this will not occur in practice.) For this reason, we will sometimes write "φ^A" to mean $\varphi^{(A)}$, where (A) is any partial oracle for A that halts everywhere. Note that φ^A can also be thought of as the partial function which outputs anything that φ could output on any partial oracle, (A), for A

This is a slight abuse of notation, but it is at least mostly justifiable, since any ordinary Turing functional can be realized as a reduction in this sense in a uniform way:

Observation 2.1.19. *Let φ_e be the eth Turing functional, in the usual sense. Then, uniformly from e, we can construct a functional ψ, in the sense of Definition 2.1.18 (formalized rigorously as an enumeration operator from inputs and outputs, in the sense of Definition 2.1.3) such that for every real X, φ^X and ψ^X halt on exactly the same inputs, and give exactly the same outputs. (In particular, $\psi^X(n)$ is never ill-defined for any X, or n.)*

Proof. In the language of this section, ψ is the functional which does exactly the same thing as φ_e, and whenever φ would query a bit of its oracle, ψ queries that same bit, and then simply waits for the oracle to give an output on that bit, refusing to go on with its construction until the oracle gives an output. If the oracle is total, then ψ clearly behaves in the same way as φ_e.

As an enumeration operator, ψ can be written as the operator which simultaneously evaluates φ_e on all oracles and all inputs, and such that whenever $\varphi_e^X(n)$ halts, then ψ enumerates a code which give the same output as $\varphi_e^X(n)$, and which uses all of the bits of X that had been queried during the computation of $\varphi_e^X(n)$.

□

Quasi-minimal generic degrees

We will discuss specific theorems that are known for the individual notions of generic reduction over the next two sections, but the remainder of this section will be devoted to discussing results that are true for all of the different notions of generic reduction. We will use the symbol $\geq_g$, but it should be understood that what we say applies equally well to any other notion of generic reduction.

Currently there is very little known about the degree structure for generic reduction. For the remainder of this chapter, much of our motivation will be derived from the following two questions.

Question 1. *Do there exist minimal degrees in the generic degrees?*

Question 2. *Do there exist minimal pairs in the generic degrees?*

A positive answer to Question 1 would provide a positive answer to Question 2, unless it were the case that there is a single nonzero generic degree that is below all other nonzero generic degrees. There is no a priori reason to dismiss this possibility, so the two questions are, at least at first glance, independent of each other.

To help us address these questions, we borrow a term from the study of enumeration reducibility, and make the following definition.

Definition 2.1.20. A generic degree $\mathbf{a}$ is *quasi-minimal* if $\mathbf{a}$ is nonzero, and if for every nonrecursive real X, the generic degree of $\mathcal{R}(X)$ is not below $\mathbf{a}$.

Then, while proving that the embedding from the Turing degrees to the generic degrees is not surjective, Jockusch and Schupp [6] actually prove the stronger result that there exists a quasi-minimal generic degree. Theorem 1.3.13 allows us to strengthen this result:

Proposition 2.1.21. *For every nonzero generic degree $\boldsymbol{a}$, there is a quasi-minimal generic degree $\boldsymbol{b}$ such that $\boldsymbol{a} \geq_g \boldsymbol{b}$.*

Proof. Let $\mathbf{a}$ be a nonzero generic degree. If $\mathbf{a}$ is quasi-minimal, then we are done. Else, choose A nonrecursive, so that the generic degree of $\mathcal{R}(A)$ is below $\mathbf{a}$. In the Turing degrees, every nonzero degree is half of a minimal pair, so choose B such that A and B form a minimal pair for Turing reducibility. By Theorem 1.3.13, choose some C, not generically computable, such that A and B can both generically compute C.

Then C generically reduces to both $\mathcal{R}(A)$, and $\mathcal{R}(B)$, and the generic degree of C must be quasi-minimal, since if there were some D such that $C \geq_g \mathcal{R}(D)$, then we would have $\mathcal{R}(A) \geq_g \mathcal{R}(D)$ and also $\mathcal{R}(B) \geq_g \mathcal{R}(D)$, and therefore $A \geq_T D$ and also $B \geq_T D$, and so D would have to be recursive, since A and B form a minimal pair. Let $\mathbf{b}$ be the generic degree of C. □

We may immediately conclude the following.

Corollary 2.1.22.

(a) If $\boldsymbol{a}$ *is a minimal generic degree, then* $\boldsymbol{a}$ *is quasi-minimal.*

(b) If $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *form a minimal pair in the generic degrees, then either* $\boldsymbol{a}$ *or* $\boldsymbol{b}$ *is quasi-minimal.*

(c) If there exist minimal pairs in the generic degrees, then there exist minimal pairs in which both of the two degrees are quasi-minimal.

Proof. A degree which is not quasi-minimal must have a quasi-minimal degree below it, and thus cannot be minimal.

If neither $\mathbf{a}$ nor $\mathbf{b}$ is quasi-minimal, then Theorem 1.3.13 provides us with a generic degree below both of them.

If $\mathbf{a}$ and $\mathbf{b}$ form a minimal pair in the generic degrees, then choose $\tilde{\boldsymbol{a}}$ and $\tilde{\boldsymbol{b}}$ to be quasi-minimal such that $\boldsymbol{a} \geq_g \tilde{\boldsymbol{a}}$ and $\boldsymbol{b} \geq_g \tilde{\boldsymbol{b}}$. Then $\tilde{\boldsymbol{a}}$ and $\tilde{\boldsymbol{b}}$ form a minimal pair in the generic degrees.

□

Unfortunately, quasi-minimal degrees are somewhat difficult to work with, because they are incapable of actually computing anything, in the usual sense. Corollary 2.1.22 shows that working with them is necessary for addressing Questions 1 and 2, so we will need to introduce some new techniques for working with them.

2.2 Uniform Generic Reducibility

In this section, we focus solely on uniform generic reduction, which is, over all, much easier to work with than nonuniform generic reduction. In some cases, these results also have corollaries concerning nonuniform generic reduction, and we point these corollaries out where able.

First, we show that uniform generic reduction is $\boldsymbol{\Pi^1_1}$-complete, then afterwards, we discuss a technique for avoiding the Π^1_1 nature of generic reduction by studying the generic degrees of density-1 sets.

$\boldsymbol{\Pi^1_1}$-completeness

Perhaps the greatest difficulty in working with generic reduction is simply that the definition is naturally a Π^1_1 definition. $A \geq_g B$ if there is a reduction procedure which generically computes B from *every* generic oracle for A. Here, we show that this universal quantifier over reals is indispensable by showing that $\geq_g$ is $\boldsymbol{\Pi^1_1}$-complete.

We accomplish this by showing that from any tree $T \subseteq \omega^\omega$, we can use the jump of that tree to uniformly find A and B such that $A \geq_g B$ if and only if T is well-founded.

For a definition and short introduction to $\boldsymbol{\Pi^1_1}$-completeness, see Appendix C. For a more thorough introduction, see [10].

Theorem 2.2.1. *There exists an algorithm which, given a tree $T \subseteq \omega^\omega$, uses T' as an oracle to compute a pair of reals A and B such that $A \geq_g B$ if and only if T is well-founded. Thus, $\geq_g$ is $\mathbf{\Pi^1_1}$-complete.*

The proof will be comprised of three parts.

In the first part, we describe the intended reduction from A to B. The reduction will have the property that an infinite path through T corresponds to a method of creating a generic oracle for A which does not generically compute B via the intended reduction. Every node of the path will be able to be translated into another drop in the density of the domain of the computation without a corresponding drop in the density of the domain of the oracle.

In the second part, we build A and B to ensure that no reduction other than the intended reduction will work. During this second part, we do not need to work effectively, but rather we have access to T'.

Finally, we verify that our construction works. If T is well-founded, this will be clear, because the intended reduction will function as it is designed to. If T is ill-founded, we will need to show that for any potential generic reduction φ, there is a generic oracle for A that either makes φ give a false answer somewhere, or that makes $\mathrm{dom}(\varphi)$ not be density-1. During this third part, we are not forced to work effectively in anything, and indeed, the generic oracles that we build would be quite difficult to compute.

Proof. **Part 1**

Let $T \subseteq \omega^\omega$.

For each $\sigma \in T$, there will be a single bit $b_\sigma \in \{0, 1\}$.

b_σ will be coded into A in a manner so that any partial oracle for A which cannot recover b_σ must have its density drop below $1 - 2^{-|\sigma|-2}$ at least once as a result.

b_σ will be coded into B in a manner so that if a computation cannot compute b_σ, then the domain of that computation will have its density drop below $\frac{1}{2}$ as a result.

The intention of this is that then, if there is an infinite path Q through T, we will be able to produce a generic oracle for A which omits b_σ for every σ on P, and which therefore cannot generically compute B, because it is missing infinitely many pieces of information about B, and each missing piece of information forces there to be another instance of the computation's domain's density dropping below $\frac{1}{2}$.

Unfortunately, this creates a problem where there could theoretically be a generic oracle for A which chooses a collection of σ's of increasing lengths from *different* paths of T, and omits each of the corresponding b_σ's, thereby potentially not being able to generically compute B even if the tree is well-founded.

For this reason, we need to also introduce a method for information to propagate down the tree: if $\sigma \prec \tau$, and b_σ is known, then it should be easy to deduce what b_τ is. That way, removing the knowledge of an entire branch will still have the original desired effect, but removing bits of information from different branches will be much more difficult than previously.

However, if we want to be able to remove information along a path, we need to make sure that our procedure for propagating information downward along T does not also cause information to propagate upward along T. Else, if Q is a path through T, $\sigma \prec \tau$, $\sigma \prec Q$, and $\tau \not\prec Q$, then b_σ would be able to be deduced from b_τ, so we would not be able to selectively remove only the information along Q from A.

Thus, for each $\sigma \in T$, for each $m < |\sigma|$, we create a procedure to deduce b_σ from $\langle b_{\sigma_0}, ..., b_{\sigma_m}\rangle$, where $\sigma_{-1} = \sigma$, and σ_{i+1} is the immediate predecessor of σ_i. This procedure is coded in a way so that if a partial oracle for A does not know the procedure, then its density must drop below $1 - 2^{|\sigma_m|+2}$ as a result. The procedure is also coded in a way so that knowing b_σ and knowing the procedure does not necessarily allow us to deduce any of the b_{σ_i}.

The actual coding is as follows:

Consider the sets P_i from Definition 1.3.5.

In B, for each $\sigma \in \omega^\omega$, choose an i in a uniform manner, and code b_σ into P_i. (If $n \in P_i$, then $n \in B \leftrightarrow b_\sigma = 1$.) If $\sigma \notin T$, then $b_\sigma = 0$.

A will be equal to $\widetilde{A} \oplus \mathcal{R}(T)$, where $\widetilde{A}$ is built as follows.

Assign half of the P_i to coding b_σ's in $\widetilde{A}$, and half of the P_i to coding deduction procedures in $\widetilde{A}$.

Then, for each $\sigma \in \omega^\omega$, choose an i in a uniform manner, and code b_σ into the last $\frac{1}{2^{|\sigma|}}$ of P_i. (If n is one of the last $2^{i-|\sigma|}$ many elements of P_i, then $n \in \widetilde{A} \leftrightarrow b_\sigma = 1$. If n is a smaller element of P_i then $n \notin \widetilde{A}$.)

Choose one P_i for each sequence $\langle \sigma, m, \tau, j, k\rangle$ such that $\sigma \in \omega^\omega, m < |\sigma|, \tau \in 2^m, j \in \{0,1\}, k \in \mathbb{N}$. Call it $P_{\sigma,m,\tau,j,k}$. Then, to deduce b_σ from the sequence $\langle b_{\sigma_0}, ..., b_{\sigma_m}\rangle$, we use the following formula.

$$b_\sigma = j \iff \exists n \exists k \; n \in \widetilde{A} \cap P_{\sigma,m,\langle b_{\sigma_0},...,b_{\sigma_m}\rangle,j,k}$$

When we actually build $\widetilde{A}$, we will ensure that for exactly one value of k, we put the last $\frac{1}{2^{|\sigma|-m-1}} \left|P_{\sigma,m,\langle b_{\sigma_0},...,b_{\sigma_m}\rangle,j,k}\right|$ many elements of $P_{\sigma,m,\langle b_{\sigma_0},...,b_{\sigma_m}\rangle,j,k}$ into $\widetilde{A}$.

For a fixed value of $\langle \sigma, m, \tau\rangle$, the set of all $P_{\sigma,m,\tau,j,k}$ is known as the *deduction procedure* coding b_σ from its $m+1$ predecessors. The deduction procedure *operates under true assumptions* if $\tau = \langle b_{\sigma_0}, ..., b_{\sigma_m}\rangle$, and it *operates under false assumptions* if $\tau = \langle b_{\sigma_0}, ..., b_{\sigma_m}\rangle$.

The idea here is that knowing $\langle b_{\sigma_0}, ..., b_{\sigma_m}\rangle$ will direct you to the correct place to look for the value of b_σ. Once you know where to look, you simply search until you find an answer. If you try to search for the value of b_σ using incorrect values for $\langle b_{\sigma_0}, ..., b_{\sigma_m}\rangle$, then you might get the correct answer, you might get the incorrect answer, and you might get no answer. Because of this, knowing b_σ gives little to no information about $\langle b_{\sigma_0}, ..., b_{\sigma_m}\rangle$. However, if we wish to remove a deduction procedure from an oracle, we only need to remove the place where it actually gives an answer, ie the last $\frac{2^i}{2^{|\sigma|-m-1}}$ many elements of P_i for some i. The size is calibrated so that removing a deduction procedure whose shortest element is τ is just as difficult as removing the knowledge of what b_τ is.

So now, given that A and B are each built in the manner just described, a generic oracle for A is able to generically compute B by the following algorithm.

Let (A) be a generic oracle for A.

To determine whether or not $n \in B$, we first determine which P_i n is in. Then we determine which b_σ is coded into that P_i. Then, since $A \geq_g \mathcal{R}(T)$, we can use (A) to determine whether or not $\sigma \in T$. If no, then $n \notin B$. If yes, then we attempt to determine the value of b_σ as follows.

We define the sentence "(A) can determine the value of b_σ." by induction on $|\sigma|$.

Let P_i be the set assigned to code b_σ in to $\widetilde{A}$. Then, if (A) gives an output on one of the last $2^{i-|\sigma|}$ many elements of P_i, then (A) can determine the value of b_σ, and that value is the value of the output that we found.

The other way that (A) can determine the value of b_σ is with our deduction procedures. If there is some $m < |\sigma|$ such that (A) can determine the values of b_τ for τ equal to one of the $m+1$ immediate predecessors of σ, and if (A) also includes the value of A in the location where the corresponding deduction procedure has a 1, then the deduction procedure allows (A) to determine the value of b_σ in the manner described when we were describing the deduction procedures.

Then, if T is well-founded, for any generic oracle (A), for A, there will only be finitely many σ such that (A) cannot determine the value of b_σ. The proof is as follows.

Let (A) be a generic oracle for A. Assume there are infinitely many σ such that (A) cannot determine the value of b_σ. Let $\widetilde{T} \subseteq \omega^\omega$ be the smallest subtree of T containing all of the σ such that (A) cannot determine the value of b_σ. Then, $\widetilde{T}$ is well-founded, because it is contained in T, which is well-founded. Also, it is infinite, by assumption. Thus, it must have at least one node where it branches infinitely. Call the first such node σ_0.

From each of those countably many branches, choose a minimal node σ such that the generic oracle cannot determine the value of b_σ. (For $i > 0$, let σ_i be en extension of the ith branch of $\widetilde{T}$ such that (A) cannot determine the value of b_{σ_i}, but such that for any τ, if $\sigma_0 \prec \tau \prec \sigma_i$, then (A) can determine the value of b_τ.)

Then, we claim that the domain of (A) must have its density drop below $1 - 2^{-|\sigma_0|-3}$ infinitely often.

The reason for this is that, for any given i, if σ_i is an immediate successor to σ_0, then the generic oracle must have its density drop below $1 - 2^{-|\sigma_0|-3}$ to not know the value of b_{σ_i} (This is by the manner in which the b_{σ_i} are directly coded into A). Otherwise, by assumption on σ_i, we know that the generic oracle *can* determine the values of the b_τ for all of the τ that satisfy $\sigma_0 \prec \tau \prec \sigma_i$. Thus, since the first of those τ has length $|\sigma_0| + 1$, the deduction procedure that allows us to determine b_{σ_i} from those b_τ is coded in a way so that if the generic oracle cannot recover that deduction procedure, then its density must drop below $1 - 2^{-|\sigma_0|-3}$.

Each of these things is coded in a different place, so the domain of the generic oracle has its density drop below $1 - 2^{-|\sigma_0|-3}$ infinitely often, so the domain is not density-1, so (A) is not a generic oracle, providing a contradiction.

Thus, if T is well-founded, any generic oracle can (uniformly) recover all but finitely many of the b_σ, and therefore all but finitely many of the bits of B. Thus, $A \geq_g B$.

Part 2

In this part, we construct $\widetilde{A}$ (and therefore A, and B, which are still defined according to the construction outlined in Part 1) in a way so that if T is ill-founded, then $A \not\geq_g B$.

If T is ill-founded, then the intended reduction will not work as a generic reduction, so the main purpose of this section will be ensuring that any reduction which "cheats" infinitely often occasionally makes mistakes, and therefore can not be used to generically reduce B to A, since a generic reduction is never allowed to give incorrect outputs. (Here, "cheating" simply means guessing at the values of b_σ without having solid evidence as to why those guesses should be correct.)

For those who like to think about constructions in terms of forcing, we will be building a generic with respect to the poset implicitly defined in Part 1, except with the additional caveats that on the "incorrect" deduction procedures, we are only allowed to encode them giving one output, giving the other output, or giving no output. Furthermore, our conditions are allowed to include restrictions of the form in the following paragraph.

One of our key techniques will be to fix a finite set of numbers, and demand that for any n in that set, any deduction procedure that operates under false assumptions about any b_σ with $|\sigma| = n$ does not produce any answers. (More formally, for any n_0 in that set, and any σ_0, σ_1, with $\sigma_0 \prec \sigma_1$, and $|\sigma_0| = n_0$, $|\sigma_1| = n_1 > n_0$, for any $m \geq n_1 - n_0 - 1$, for any τ such that $\tau(n_1 - n_0 - 1) \neq b_{\sigma_0}$, for any j, and k, $P_{\sigma_1,m,\tau,j,k} \cap \widetilde{A}$ is empty.)

The actual construction is as follows.

At the beginning of stage s, there is some number $f(s)$ such that we have determined the values of b_σ for every σ such that $|\sigma| < f(s)$, and for no other σ. For every σ with $|\sigma| < f(s)$, and every deduction procedure for computing b_σ, we have determined the values of $\widetilde{A}$ on the entire deduction procedure. (That is to say, if $|\sigma| < f(s)$, then we have determined whether or not $n \in \widetilde{A}$ for every n in any $P_{\sigma,m,\tau,j,k}$.) We also have some finite set of numbers n such that we demand that for any n in that set, any deduction procedure that operates under false assumptions about any b_σ with $|\sigma| = n$ does not produce any answers. We have not determined the values of $\widetilde{A}$ on any other deduction procedures.

At this point, we have one single functional, ϕ_s, that we need to diagonalize against. This means that we either must ensure that there will be some generic oracle (A) for A such that $\phi_s^{(A)}$ does not have density-1 domain, or we must ensure that there is some generic oracle (A) for A such that $\varphi_s^{(A)}$ incorrectly computes B at some number.

The first question that we ask is whether there any way of extending our definition of $\widetilde{A}$ to make φ_s^A produce an incorrect computation for B. (Again, we remind the reader that T is fixed, and B depends entirely on A, so determining the value of $\widetilde{A}$ also determines the values of A and B, and so determines whether or not φ_s^A produces any incorrect computations for B. Also, generic computations are defined in a way so that having more information about the oracle always produces more outputs, and never produces different outputs, and thus, if any generic oracle for A produces incorrect results, then the full oracle for A produces incorrect results.)

If φ_s^A can be made to produce any incorrect computation for B, then we make that extension, and this guarantees that when we are done constructing $\widetilde{A}$, (and therefore A and B,) it will not be true that $A \geq_g B$. After that, we extend $\widetilde{A}$ arbitrarily in order to meet the hypotheses on what the construction should look like at the beginning of a stage. (This involves finding the largest number c such that b_σ is defined for a σ with $|\sigma| = c$, or such that some deduction procedure for such a σ is defined somewhere, and extending the definition of $\widetilde{A}$ arbitrarily to all other b_τ's and deduction procedures for b_τ's of equal or lesser length.)

If φ_s^A cannot be made to produce any incorrect computations for B, then we extend our definition of $\widetilde{A}$ such that for every σ with $|\sigma| = f(s)$, $b_\sigma = 0$. More importantly, we restrict every deduction procedure that computes such a b_σ and that operates under false assumptions to not give any answers. (Ie, to be empty.) We allow all the correct deduction procedures to give correct answers immediately. Finally, we insist that for the rest of the construction, for every deduction procedure that operates under a false assumption about the value of b_σ for any σ with $|\sigma| = f(s)$, that deduction procedure does not give any answers.

This completes the construction. (The second option will certainly happen infinitely often, and so $\widetilde{A}$ will be defined everywhere after ω many steps.)

Part 3

Finally, we prove that if T is ill-founded, then $A \not\geq_g B$. To do this, we must demonstrate that for any φ that was not able to be extended to make a false claim about B, there exists a generic oracle, (A), for A such that $\varphi^{(A)}$ does not have density-1 domain. (At the end of Part 1, we verified that if T is well-founded, then $A \geq_g B$. Also, if φ could have been able to be extended to make a false claim about B, we would have done that, and then it would certainly not witness $A \geq_g B$.)

We remind the reader that in this part of the proof, we are allowed to work omnisciently, and in particular we will be allowed to know every choice that was made in Part 2, as well as knowing an example of an infinite path through T.

Assume T is ill-founded. Let Q be an infinite path through T. Assume that at stage s of the construction φ_s^A could not have been made to produce any incorrect computations for B.

Let $s_0 = s$, and for each $i > 0$, let s_i be the ith stage t after stage s such that φ_t^A was not able to be made to produce any incorrect computations for B at stage t. Let σ_i be the initial segment of Q such that $|\sigma_i| = s_i$

Now, we define (A) to be the oracle for A which does not give answers on the coding locations of any of the b_{σ_i}, and which also does not include the answers (the 1s) from any of the deduction procedures used for deducing b_{σ_0}, or from any of the deduction procedures used for deducing b_{σ_i}, unless those deduction procedures have sufficiently large m values that they depend on $b_{\sigma_{i-1}}$.

Then we claim that (A) is a generic oracle for A, and that $\varphi_s^{(A)}$ does not have density-1 domain.

To show that (A) is a generic oracle for A, we show that the set of places where (A) does not give answers is density-0.

There are finitely many (in fact, at most one) σ_i of each possible length, and so the union of all of the coding locations of all of the b_{σ_i} is density-0. (This is by Lemma 1.3.7, because each b_{σ_i} is coded into a smaller portion of its corresponding P_j than the previous one.) Also, for each length n, there is at most one deduction procedure whose answers we erased whose shortest queried string has length n. This is because only the correct deduction procedures give any answers that need to be erased, and because we only erase deduction procedures that do not require knowledge of the previous b_{σ_i}. Thus, again, the union of all the the erased answers from deduction procedures has density-0.

Finally, we show that $\varphi_s^{(A)}$ does not give outputs in any of the locations where the b_{σ_i} are coded in B.

The proof of this fact is that, for any i, any finite subset of the information in (A) is a partial oracle that could be extended to a partial oracle for a different set A_1 which would also be consistent with the requirements imposed at the beginning of stage s of the construction, and such that the value of b_{σ_i} in A_1 was different from the value of b_{σ_i} in A. (This statement will be proved by induction shortly) Therefore, if $\varphi_s^{(A)}$ gives any outputs on any of the the b_{σ_i}, then at stage s, we would have been able to extend our condition on A to a condition specifying enough of A_1 to cause φ_s to produce an incorrect computation for B, contradicting our assumption on φ_s.

We conclude our proof by proving by induction on i, that any finite subset of the information in (A) is a partial oracle that could be extended to a partial oracle for a different set A_1 which would also be consistent with the requirements imposed at the beginning of stage s of the construction, and such that the value of b_{σ_i} in A_1 was different from the value of b_{σ_i} in A.

Recall that by the construction of $\widetilde{A}$, and by the assumption on σ_i, for every i, $b_{\sigma_i} = 0$

Case 1: $i = 0$

Assume that we have seen a finite number of the pieces of information in (A).

Then we have not seen any of the locations where b_{σ_0} is coded, and we have also seen nothing but 0's from all of the deduction procedures for computing b_{σ_0}. At stage s, the value of b_{σ_0} had not yet been decided, and there were no conditions yet on the deduction procedures for computing b_{σ_0}, except for some requirements that certain deduction procedures that operated under false assumptions were not allowed to give any outputs. Furthermore, none of the deduction procedures operating under false assumptions about b_{σ_0} ever give answers. Therefore, it would be consistent with what we have seen so far of (A) and with the requirements imposed at the beginning of stage s to have b_{σ_0} equal to 1, and then to fill in A_1 with $b_{\sigma_0} = 1$, and with the deduction procedures that compute b_{σ_0} having 1's in the correct locations in the next coding sets that we have not yet looked at. Since none of the deduction procedures that operated under the assumption $b_{\sigma_0} = 1$ have given answers in the finite amount of (A) that we have seen, we may freely extend them in A_1 to give correct outputs in A_1. (Notice that this is consistent with the conditions imposed on the construction at the beginning of stage s, because the condition that says that those deduction procedures must never give outputs was imposed at the end of stage s.)

Case 2: $i > 0$

Assume again that we have seen a finite number of the pieces of information in (A).

Then, again, we have not seen any of the locations where b_{σ_i} is coded, and we have also seen nothing but 0's from all except a few of the deduction procedures for computing b_{σ_i}. Of these deduction procedures, the only ones from which we have seen any 1's are the ones which operate under true assumptions, and which also depend on the value of $b_{\sigma_{i-1}}$. By induction, it is consistent (both with what we have seen of (A), and with the condition on the construction at stage s) that $b_{\sigma_{i-1}}$ could have the opposite value from its actual value. Thus it would be consistent to fill in A_1 to have the incorrect value for $b_{\sigma_{i-1}}$, and then to also have the incorrect value for b_{σ_i}, and then to have the "relevant" deduction procedure then place 1's into the next relevant location. (Here, relevant means operating under the assumptions that are true of A_1, but potentially false of A_0.) Again, this does not contradict any of our requirements, since none of the deduction procedures which use $b_{\sigma_i} = 1$ have given any outputs yet, so the ones which operate under assumptions which are correct in A_1 can still be made to give outputs which are correct in A_1.

This concludes our proof of $\mathbf{\Pi^1_1}$-completeness of $\geq_g$.

$\square$

We briefly mention two things about this proof.

First of all, the fact that a deduction procedure operating under false assumptions about b_{σ_i} never gives incorrect answers is very important in order to ensure that it is consistent with any finite oracle that b_{σ_i} has the opposite value. However, it is also important that it never gives correct answers, because it might be possible to deduce the value of $b_{\sigma_{i+1}}$ without knowing the value of b_{σ_i}. (And to avoid that, we might be forced to remove too much information from our oracle (A).) It is also important that sometimes, incorrect deduction procedures can give correct answers, because otherwise, seeing a deduction procedure give a correct answer would tell you that all of its assumptions were correct, and it is also important that sometimes incorrect deduction procedures give incorrect answers, because otherwise, seeing any answer from any deduction procedure would be sufficient to know that the answer was correct. By using all three sorts of deductions (incorrect, correct, and nonresponsive), we have enough leeway to force the opponent to not give answers if he is clever enough to avoid making mistakes.

The second thing is that this proof is almost a proof that nonuniform generic reduction is $\mathbf{\Pi^1_1}$ complete, but there are two obstructions. First of all, we use almost the same generic oracle for every φ, except that φ_s could potentially "know" about the infinitary requirements imposed before stage s, (since we cannot break those requirements in order to make φ_s give an incorrect answer,) so for larger values of s, we need to go deeper into T before we begin removing information from (A). We cannot intersect all of these oracles, because that would involve removing too many deduction procedures that used bits all the way from the root of the tree.

The second obstruction is that even if we were able to overcome this, we would still need to somehow prove that none of the φ that gave incorrect answers would be able to produce a generic computation of B from our density-1 oracle. (In nonuniform generic reduction, giving incorrect answers on one oracle does not disqualify you from being a legitimate generic reduction on some other oracle, and we would need to create one single oracle from which no φ would be able to produce a generic computation.)

This seems quite difficult, which is somewhat strange, given that nonuniform generic reduction is naturally defined in a Π^1_1 manner (For every generic oracle, there exists a reduction such that... etc) whereas uniform generic reduction is naturally defined with a leading existential quantifier over naturals (There exists a reduction such that for every generic oracle, etc.) Nonuniform generic reduction is probably also $\boldsymbol{\Pi^1_1}$-complete (it would be very surprising if this seemingly much more complicated notion of reduction was actually less complicated in the definability hierarchy), but unfortunately, it is sufficiently difficult to work with that we do not know how to prove that it is $\boldsymbol{\Pi^1_1}$-complete.

Densiy-1 degrees

When working with generic computation and generic reduction, in general, one needs to balance working with what the halting sets of computations should be, and what the outputs of those computations on their halting sets should be. In some sense, the work with the halting sets is the new concept that we explore in generic computation, and the outputs can be dealt with in similar ways as they are dealt with in ordinary Turing reduction. To isolate and better understand the role that halting sets have in the theory of generic reduction, we consider the generic degrees of density-1 sets.

Definition 2.2.2. A generic degree $\mathbf{a}$ is *density-1* if there exists a real $A \in \mathbf{a}$ such that A is density-1.

As it turns out, a generic degree having a density-1 real in it is equivalent to it having a coarsely computable real in it. (From Definition 1.1.12.)

Proposition 2.2.3. *A generic degree $\boldsymbol{a}$ is density-1 if and only if there exists a coarsely computable real $A \in \boldsymbol{a}$, ie a real $A \in \boldsymbol{a}$ and a recursive real B, such that $\{n \mid A(n) = B(n)\}$ is density-1.*

Proof. Certainly, if a generic degree is density-1, then the second half holds by letting A be a density-1 real in $\mathbf{a}$, and by letting $B = \mathbb{N}$.

The converse follows directly from the following lemma. □

Lemma 2.2.4. *Let A, and B be reals, and assume that B is recursive, and that $\{n \mid A(n) = B(n)\}$ is density-1.*

Then $\{n \mid A(n) = B(n)\} \equiv_g A$.

Proof. $A \geq_g \{n \mid A(n) = B(n)\}$ via the algorithm φ where $\varphi(n) = 1$ for any n such that $A(n) = B(n)$. Since B is recursive, and since any generic oracle gives density-1 many of the bits of A, and since the intersection of two density-1 sets is density-1, any generic oracle for A will find density-1 many locations where A and B agree.

Conversely, $\{n \mid A(n) = B(n)\} \geq_g A$ via $\varphi(n) = B(n)$ for any n such that $A(n) = B(n)$. The oracle tells φ where to halt, and a computation of B on that input tells φ what output to give. The domain is density-1 again because the intersection of two density-1 sets is density-1. □

The two most fundamental facts for this subsection are that the intersection of two density-1 sets is density-1, and that when working with density-1 sets, we may work entirely with oracles and computations that only output ones. The proof of this first fact is briefly sketched after Definition 1.1.1, and the second fact is just a generalization of Observation 1.1.11, but we state and prove both for the sake of completeness.

Observation 2.2.5. *Let A and B be reals. Then $A \cap B$ is density-1 if and only if both A and B are density-1.*

Proof. If $A \cap B$ is density-1, then each of A and B is density-1 because it is a superset of a density-1 set.

Conversely, fix some $\epsilon > 0$. We must prove that there is some m such that $\forall n > m \ \frac{|(A \cap B)\restriction n|}{n} > 1 - \epsilon$.

If A and B are each density-1, then choose m such that for every $n > m$, $\frac{|A\restriction n|}{n} > 1 - \frac{\epsilon}{2}$ and $\frac{|A\restriction n|}{n} > 1 - \frac{\epsilon}{2}$. This m suffices for our purposes. □

Observation 2.2.6. *Let A and B be density-1 reals. Then $A \geq_g B$ if and only if there exists a Turing functional φ which never produces 0 as an output, such that for any generic oracle (A) for A, if every output of (A) is a 1, then $\varphi^{(A)}$ is a generic computation of B.*

Proof. Let A and B be density-1 reals.

Assume $A \geq_g B$. Fix φ such that for any generic oracle (A) for A, $\varphi^{(A)}$ is a generic computation of B. If we modify φ to not halt when it otherwise would halt and output 0, then it will still be true that for any (A), $\varphi^{(A)}$ is a generic computation of B, because the outputs that it gives will still be correct, and the domains will still be density-1, since for each (A), the domain of $\varphi^{(A)}$ will be the intersection of B with the previous domain of $\varphi^{(A)}$.

Since this reduction works for every generic oracle, in particular, it also works for generic oracles whose domain is contained in A.

Conversely, assume there exists a φ which generically computes B from any generic oracle for A which only outputs 1s. Then it can be modified to generically compute B from any generic oracle (A) for A because A is density-1, (A) must contain density-1 many elements of A in its domain, and φ may be modified to ignore the zeroes, and to give exactly the same outputs as it would have given if those zeroes were not in the domain of (A).

□

Observation 2.2.6 is usually only used implicitly, in that it was useful in narrowing the search for reductions, but not in writing the proof. It will occasionally be used explicitly. More importantly than that, however, is the philosophy implicit in the observation: that generic descriptions of density-1 sets should be thought of as density-1 subsets of those sets.

Observation 2.2.5 is used explicitly ad nauseum when working with density-1 degrees, and references to it will frequently be suppressed to decrease proof length.

We apply Observation 2.2.5 to prove two results which give an idea of why density-1 degrees are easier to work with than other generic degrees.

Lemma 2.2.7. *Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be density-1 generic degrees. Then $\boldsymbol{a} \geq_g \boldsymbol{b}$ if and only if $\exists A \in \boldsymbol{a}\ \exists B \in \boldsymbol{b}$ such that A and B are both density-1, and $A \subseteq B$.*

Proof. Assume $\exists A \in \mathbf{a}\ \exists B \in \mathbf{b}(A \subseteq B)$, with A and B both density-1. Then $A \geq_g B$ via the algorithm φ where $\varphi(n) = 1$ if $n \in A$. This algorithm only gives correct outputs because $A \subseteq B$, and it halts on the intersection of A with the domain of the generic oracle for A, which is density-1.

Conversely, assume $\mathbf{a} \geq_g \mathbf{b}$, and let $A_0 \in \mathbf{a}, B_0 \in \mathbf{b}$, with A_0, B_0 both density-1. Let $A = A_0 \cap B_0$, and let $B = B_0$. By definition, $A \subseteq B$. Also, A_0 is density-1 by Observation 2.2.5. So it remains to show that $A \equiv_g A_0$.

$A \geq_g A_0$ because $A \subseteq A_0$ and A is density-1. Conversely, $A_0 \geq_g A$ because a generic oracle for A_0 can be used to generically compute B_0, and we may then restrict the domain of this computation to only give outputs on elements of A_0. This has density-1 domain because the domain is the intersection of A_0 with the domain of the generic computation of B_0, and it always correctly computes A because it correctly computes B, and for any $n \in A_0$, $A(n) = B(n)$. □

Lemma 2.2.8. *Let A and B be density-1 reals. Then $A \cap B \equiv_g A \oplus B$.*

Proof. By Lemma 2.2.7, $A \cap B \geq_g A \oplus B$ (since both A and B generically reduce to it.)

Conversely, $A \oplus B \geq_g A \cap B$ by the algorithm φ where $\varphi(n) = 1$ if $2n$ and $2n+1$ are both in $A \oplus B$. (If we would like, we may also have $\varphi(n) = 0$ if either $2n$ or $2n+1$ is not in $A \oplus B$, this would still be a correct generic reduction, but it is against the philosophy implicit in Observation 2.2.6.) The domain is density-1 by the usual proof that $A \oplus B \geq_g A$, and $A \oplus B \geq_g B$. (See Lemma 2.1.17.)

□

We use Lemma 2.2.7 to prove one of our more amusing results: that the density-1 sets are dense!

Proposition 2.2.9. *Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be density-1 generic degrees. Assume $\boldsymbol{a} >_g \boldsymbol{b}$. Then there exists a density-1 degree $\boldsymbol{c}$ such that $\boldsymbol{a} >_g \boldsymbol{c} >_g \boldsymbol{b}$.*

Proof. Use Lemma 2.2.7 to fix density-1 sets $A \subseteq B$ in $\mathbf{a}$ and $\mathbf{b}$ respectively.

We build a real C such that $A \subseteq C \subseteq B$, which guarantees that C is density-1 and that $A \geq_g C \geq_g B$. The main difficulty of the construction will be ensuring that $C \not\geq_g A$, and $B \not\geq_g C$.

We caution the reader to notice that it is not necessarily important to ensure that $A \geq_T C$, and indeed, this will probably not be the case.

The basic idea of the proof is that C will copy A until it forces one instance of B not computing it, then it will copy B until it forces one instance of it not computing A, then it switches back to copying A.

At stage $2e$, we have some finite approximation σ_{2e} to C, and we wish to extend it so that C does not generically reduce to B via φ_e. The first thing that we ask is whether it is true that for every generic oracle (B), for B, $\varphi_e^{(B)}$ is a generic computation of B which never outputs any 0s. If not, then we do not have to do anything, since any generic computation of C must also be a generic computation of B. (C will be density-1 when we build it since it will contain A, so by Observation 2.2.6, we may assume that any generic computation of C outputs density-1 many 1s, and no 0s, and C will be contained in B, so that generic computation would also be a generic computation of B.)

C will be density-1 when we build it (it will contain A), by Observation 2.2.6, we may assume that any generic computation of C outputs density-1 many 1s, and no 0s, and C will be contained in B, so any such generic computation must have domain contained in B, and thus be a generic computation of B.

If yes, then there must be some number n such that $\varphi_e^B(n) = 1$, but $n \notin A$. Otherwise, $B \geq_g A$ via φ_e. (Recall that generic reductions are defined in a way so that having less information about the oracle never results in more outputs, so if $\varphi_e^B \subseteq A$, then for every (B), $\{n \mid \varphi_e^{(B)}(n) = 1\}$ is a density-1 subset of A, and so $\varphi_e^{(B)}$ is a generic computation of A.)

By the usual argument, there must be infinitely many such n (because otherwise we could modify φ_e to not halt on those n.) So we can choose one such n that is larger than $|\sigma_{2e}|$, and extend our approximation to C to be equal to A up to and including that n.

This ensures that it is not true that $B \geq_g C$ via φ_e.

At stage $2e+1$, we have some finite approximation σ_{2e+1} to C, and we wish to extend it so that A does not generically reduce to C via φ_e. The first thing that we ask is whether it is true that for every generic oracle (A), for A, with $\mathrm{dom}((A)) \subseteq A$, $\varphi_e^{(A)}$ is a generic computation of A. If not, then we do not have to do anything: C will contain A, so every such generic oracle for A will, in fact, also be a generic oracle for C. Thus if any of the $\varphi_e^{(A)}$ is not a generic computation of A, then A will definitely not reduce to C via φ_e.

If yes, then it must be true that for some $n \notin A$, $\varphi_e^B(n) = 1$. Otherwise, $B \geq_g A$ because for any (B), $\varphi_e^{(B)}$ never gives incorrect outputs about A. (We implicitly use Observation 2.2.6 in assuming that φ_e can only make mistakes by outputting 1s when it is not supposed to. Notice that any generic oracle for B actually is a superset of some generic oracle for A, and so for any (B), the domain of $\varphi_e^{(B)}$ is density-1.)

Similarly, if we let B_1 be the real which agrees with σ_{2e+1} up to $|\sigma_{2e+1}|$, and which agrees with B after that, then there must also be some $n \notin A$, $\varphi_e^{B_1}B(n) = 1$ because otherwise

$B_1 \geq_g A$, which is not possible, since it is a finite modification of B. We extend our approximation to C to be equal to B until we have copied enough to make $\varphi_e^C B(n) = 1$ for some $n \notin A$.

This ensures that it is not true that $C \geq_g A$ via φ_e.

Then, after countably many stages, we have ensured that there is no φ witnessing either $B \geq_g C$, or $C \geq_g A$. Hence, because $A \subseteq C \subseteq B$, we have that that $A >_g C >_g B$. Let **c** be the generic degree of C.

□

We can strengthen this result to split A over B:

Proposition 2.2.10. *Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be density-1 generic degrees. Assume $\boldsymbol{a} >_g \boldsymbol{b}$. Then there exist density-1 degrees $\mathbf{c}$ and $\mathbf{d}$ such that $\boldsymbol{a} >_g \boldsymbol{c} >_g \boldsymbol{b}$, $\boldsymbol{a} >_g \boldsymbol{c} >_g \boldsymbol{b}$, and $\boldsymbol{c} \oplus \boldsymbol{d} = \boldsymbol{a}$.*

Proof. The basic idea of this proof will be that we will mimic the previous construction, but we will ensure that $C \cap D = A$, which will ensure that $C \oplus D \equiv_g A$ by Lemma 2.2.8. We may also, if we desire, ensure that $C \cup D = B$, which we will do, just for symmetry, but this does not ensure that **b** is the infimum of **c** and **d** in the generic degrees.

We will assume familiarity with the proof of 2.2.9.

As before, fix density-1 sets $A \subseteq B$ in **a** and **b** respectively.

At stage $2e$, we have some finite approximations σ_{2e} to C, and τ_{2e} to D, and we wish to extend them so that C does not generically reduce to B via φ_e, and so that A does not generically reduce to D via φ_e. (Notice here that we are satisfying requirement $2e$ for C, but requirement $2e+1$ for D.)

We accomplish this by having C copy A until φ_e^B incorrectly computes some bit of C. We simultaneously have D copy B until φ_e^D incorrectly computes some bit of A. (If one of the strategies satisfies its objective before the other one does, then it continues to copy the set that it is copying until the other strategy has satisfied its own objective.) If one of the strategies does not need to act, then both strategies only wait for the one that needs to act to satisfy its objective. If neither strategy needs to act, then we simply move on to stage $2e+1$. (Again, this cannot be done uniformly, but that is fine, because we do not need either C or D to be computable from A, but rather just for them both to be generically computable from A, which will be satisfied by both of them containing A.)

At stage $2e+1$, we have C copy B and D copy A in a manner analogous to what we did at stage $2e$.

At the end of the construction, we have that $C \cap D = A$, and so $C \oplus D \equiv_g A$. We have also forced that $C \not\geq_g A, D \not\geq_g A, B \not\geq_g C$, and $B \not\geq_g D$. So we may let **c**, and **d** be the generic degrees of C, and D, respectively.

□

Now, that we have established some techniques for working with density-1 degrees, we discuss what is known about how the density-1 degrees embed in the generic degrees as a whole.

Proposition 2.2.11. *Let $\boldsymbol{a}$ be any non-quasi-minimal generic degree. Then there is a quasi-minimal density-1 generic degree, $\boldsymbol{b}$, such that $\boldsymbol{a} >_g \boldsymbol{b}$.*

Proof. The proof of Proposition 2.1.21 also proves Proposition 2.2.11, since all of the sets that are generically computed in the proof of Theorem 1.3.13 are density-1. (We find A such that $\mathbf{a} \geq_g \mathcal{R}(A) >_g 0$. We then find a real C so that A and C form a minimal pair in the Turing degrees, and then since they do not form a minimal pair for generic computation, we may let $\mathbf{b}$ be the generic degree of the real that both of them can generically compute.) □

Notice that Proposition 2.1.21 is proved for all generic degrees (indeed, it is trivially true for quasi-minimal generic degrees), however, our proof of Proposition 2.2.11 only works for non-quasi-minimal generic degrees.

We also show that in some sense, the density-1 degrees go reasonably high up in the generic degrees.

Proposition 2.2.12. *Let A be any nonrecursive real which can be uniformly computed from a sufficiently fast growing function. Then there is a density-1 real B such that $B \geq_g \mathcal{R}(\mathcal{A})$.*

We clarify that when we say that A can be uniformly computed from a sufficiently fast growing function, we mean that there is a single increasing function f, and a single Turing functional φ such that for any g such that $\forall n\, g(n) \geq f(n)$, φ^g computes A. (This hypothesis holds for precisely the hyperarithmetic reals. See Appendix D for a definition of the hyperarithmetic reals, and a proof of this fact.)

We also mention that for this proof, we use uniform time-dependent generic reduction, because the time-independent version of the proof is more difficult to make precise.

Proof. Let f be a function so that any faster growing function computes A. Let φ be the reduction that witnesses this.

Let B be any density-1 real such that for every n, and m, if $n < f(m)$, then $\frac{|B\restriction n|}{n} < 1-2^m$. For any increasing f, there is such a B because for any fixed m_0, the density of B will eventually be allowed to go above $1-2^{m_0}$ (as soon as n is larger than $f(m_0)$,) and so B can satisfy the "slow growing" requirement while still having its density go to 1 eventually.

Then, from any time-dependent generic oracle (B) for B, we can define a function g, where $g(m)$ is the first number n such that we see that $\frac{|B\cap dom((B))\restriction n|}{n} \geq 1-2^m$. $g(m)$ is defined, because the density of (B) must go to 1, so there must be some least n_0 such that $\frac{|B\cap dom((B))\restriction n_0|}{n_0} \geq 1-2^m$, and so there must be some $n \geq n_0$ where (B) first shows this to happen.

$B \cap dom((B)) \subseteq B$, so we have that for every m, $g(m) \geq f(m)$, so we may compute A, and therefore compute (and generically compute) $\mathcal{R}(A)$ via $n \in A \leftrightarrow \varphi^g(n) = 1$.

□

We mention that in the previous proof, g is computed uniformly from the time-dependent generic oracle, but two different time-dependent generic oracles that gave exactly the same outputs might give different g's, depending on the time dependence. By Proposition 2.1.10, this proof also proves the time-independent version, but if we wanted to do the time-independent proof directly, we would need to apply φ to all apparently consistent values for g, and give outputs whenever φ halted on any of them. So no single g would be computed as an intermediate step, but A would still be computed uniformly in the end.

On the other hand, we now show that the density-1 degrees can only bound countably many Turing degrees.

Proposition 2.2.13. *There are at most countably many reals A such that there is a density-1 real B such that $B \geq_g \mathcal{R}(A)$.*

The key idea of this proof is that for any fixed φ, there can be at most one A such that there exists a density-1 real B such that $B \geq_g \mathcal{R}(A)$ via φ. This is because given any two density-1 reals, there is a generic oracle that is a generic oracle for both of them.

Proof. Fix a Turing functional φ. Assume that $B_0 \geq_g \mathcal{R}(A_0)$ via φ, and also that $B_1 \geq_g \mathcal{R}(A_1)$ via φ, for some density-1 reals B_0, and B_1, and for some reals A_0, and A_1.

Then, let (B) be the generic oracle which outputs a 1 if and only if $n \in B_0 \cap B_1$, and which never outputs a 0.

Then $\varphi^{(B)}$ is a generic computation of $\mathcal{R}(A_0)$, because (B) is a generic oracle for B_0, and likewise, it is a generic computation of $\mathcal{R}(A_1)$, because (B) is a generic oracle for B_1.

Thus, $A_0 = A_1$, because otherwise $\mathcal{R}(A_0)$ and $\mathcal{R}(A_1)$ would differ on a set of positive density, and so it wouldn't be possible for one single generic computation to generically compute both of them.

Thus, for each φ, there is at most one $\mathcal{R}(A)$ that generically reduces to some density-1 real via φ, so since there are only countably many Turing reductions, there can be at most countably many $\mathcal{R}(A)$s that have a density-1 degree above them.

□

We combine the ideas from these two proofs to prove the converse to Proposition 2.2.12, demonstrating a characterization of the hyperarithmetic reals in terms of generic reducibility, and density-1 sets.

Theorem 2.2.14. *A real A is hyperarithmetic if and only if there is a density-1 real B such that $B \geq_g \mathcal{R}(\mathcal{A})$.*

The basic idea of the construction is that we use B to generate a function such that from any faster growing function, one can generate a binary tree of density-1 oracles that includes B. (The function generated is not the same as the function from Proposition 2.2.12, since this function's purpose is in some sense dual to the purpose of the previous function.)

Once we have this class of oracles, we have them engage in a process that can be visualized as them voting in pairs until they can elect one leader who is strong-willed enough to make them vote unanimously. B is able to make them vote unanimously, so eventually they will find such a leader, and B also is not corruptible, so when they find such a leader, even if that leader is not B, that leader will have B's support, and so in particular that leader will lead the oracles to the correct conclusion.

Proof. For this proof, we use the characterization of the hyperarithmetic reals as the reals for which there exists a function such that the real can be uniformly computed from any faster growing function, as in the comment after Proposition 2.2.12. (See Theorem D.0.25, in Appendix D.)

The forward direction of the theorem follows directly from Proposition 2.2.12, so we only prove the other direction.

Let A be a real, B a density-1 real, and φ a Turing functional such that $B \geq_g \mathcal{R}(A)$ via φ.

Then, let f be the function where $f(m)$ is the smallest number such that $\forall n > f(m)$, $\frac{\#(B\upharpoonright n)}{n} > 1 - 2^m$. Then we claim that from any g that dominates f, we can uniformly compute A.

The basic idea of the construction is that from any such g, we can get a lower bound on the rate at which the density of B goes to 1. We then consider all density-1 oracles whose density goes to 1 at at least that rate (this bound allows us to build a tree of possibilities for B that includes only paths which are density-1,) and we use the techniques of the proof of Proposition 2.2.13 to eliminate the incorrect answers.

Let g be a function such that for all m, $g(m) \geq f(m)$. By replacing g with a faster growing function if necessary, we may assume that g is an increasing function.

(Replace g with $\tilde{g}$, where $\tilde{g}(m) = max(g(m), \tilde{g}(m-1)+1)$.)

Define $T_g \subseteq 2^{<\omega}$ to be the tree where $\sigma \in T_g$ if and only if $\forall n, m$, if $n > g(m)$, and if $n \leq |\sigma|$, then $\frac{\#\{k<n \mid \sigma(k)=1\}}{n} > 1 - 2^m$. The only relevant facts about T_g are that every path through T_g is density-1, B is a path through T_g, and that T_g is uniformly recursive in g.

The first fact is because g is used to provide a lower bound on the rate at which the density of a path must go to 1, and that paths through T_g all respect that lower bound. The second fact is because, by definition B is a path through T_f, and faster growing functions provide larger trees, not smaller ones. The third fact is because the definition is uniform in g (and the apparently unbounded quantifiers over n and m are bounded by $g(m) < n \leq |\sigma|$.)

To compute whether or not $n \in A$, we search for a real X_0 such that for every X that looks like it might be a path through T, if we let $Y_X = X_0 \cap X$, and let (Y_X) be the partial oracle for Y_X which only halts on the elements of Y_X, then for some odd value of k, $\varphi^{(Y_X)}(k2^n)$ halts, and such that all of those computations (ranging over different reals X) halt and give the same answer. Then $n \in A$ if and only if that answer is "1."

Such a real exists because B is such a real. (If we let $X_0 = B$, then for every X, either X can eventually be shown to not be a path through T, or (Y_X) is a generic oracle for B, and so $\varphi^{(Y_X)}$ must be a generic computation of $\mathcal{R}(\mathcal{A})$, and thus it must halt on some $k2^n$. This is observed at some finite stage by the usual compactness argument.)

Furthermore, the answer given when X_0 is found must be correct, because B is a path through T_g, and so is always one of the eligible values for X, so for any X_0, (Y_B) is a partial oracle for B. ((Y_B) is only a generic oracle for B if X_0 is density-1, but that does not matter.)

Any partial oracle for B can be extended to a generic oracle for B by simply giving more outputs, and so $\varphi^{(Y_B)}$ cannot give any incorrect outputs for $\mathcal{R}(\mathcal{A})$.

Thus, once again, intuitively speaking, B is smart enough to force every X to give the correct vote, so a consensus must eventually be reached, and no X_0 is able to force B to vote incorrectly, so any reached consensus must be a correct one. g is only used in order to build a population of density-1 sets that includes B as a member.

□

We now present a major open question concerning density-1 sets, and then discuss the corollaries that a resolution to the question could have.

Question 3. *Is it true that for every nonzero generic degree $\boldsymbol{a}$ there exists a nonzero density-1 generic degree $\boldsymbol{b}$ such that $\boldsymbol{b} \leq_g \boldsymbol{a}$?*

The particularly interesting thing about this question is that a resolution to it would allow us to either resolve Question 1 or Question 2. If the answer is positive, then it allows us to prove that there are no minimal degrees in the generic degrees, because the density-1 generic degrees are dense, and 0 is a density-1 generic degree. However, if the answer is negative, then we can deduce that there exist minimal pairs in the generic degrees, by an argument that is analogous to the argument from Proposition 1.3.3, because an example of a degree that does not bound a density-1 degree would be an analogue to a nonrecursive set such that every density-1 set that it could enumerate would contain a density-1 r.e. subset.

Proposition 2.2.15. *If for every nonzero generic degree $\boldsymbol{a}$ there exists a nonzero density-1 generic degree $\boldsymbol{b}$ such that $\boldsymbol{b} \leq_g \boldsymbol{a}$, then there are no minimal degrees in the generic degrees.*

Proof. Let $\mathbf{a}$ be a generic degree, and assume that there exists a nonzero density-1 generic degree $\mathbf{b}$ such that $\mathbf{b} \leq_g \mathbf{a}$.

Then, 0 is a density-1 generic degree (because $\mathbb{N}$ is generically computable, and density-1), and so by Proposition 2.2.9, there exists a density-1 generic degree $\mathbf{c}$, such that $\mathbf{b} >_g \mathbf{c} >_g 0$. Thus, $\mathbf{a} >_g \mathbf{c} >_g 0$, and so $\mathbf{a}$ is not a minimal generic degree.

□

Proposition 2.2.16. *If there exists a nonzero generic degree $\boldsymbol{a}$ such that there is no nonzero density-1 generic degree $\boldsymbol{b}$ with $\boldsymbol{b} \leq_g \boldsymbol{a}$, then $\boldsymbol{a}$ is half of a minimal pair in the generic degrees.*

This proof is the analogue of the proof of Proposition 1.3.3.

Proof. Let $A \in \mathbf{a}$. We will build a real C such that $\mathcal{R}(C)$ and A form a minimal pair for generic reduction, or in other words, so that if $A \geq_g D$, and $C \rightarrow_g D$, then D is generically computable.

We build C by finite approximation.

We have one stage for each e, and one stage for each $\langle i, j \rangle$.

At stage e, we ensure that C is not computed by φ_e in the usual manner.

At stage $\langle i, j\rangle$, we have an approximation τ for C, and we need to ensure that if D generically reduces to A via φ_i, and if D is generically computable from C via φ_j, then D is generically computable.

(As a reminder, being generically computable from C is equivalent to being generically reducible to $\mathcal{R}(C)$. Also, being generically equivalent to 0 equivalent to being generically computable.)

Then, the first thing that we ask is whether there is any extension $\tilde{\tau}$ of τ such that for some n, $\varphi_i^A(n) \neq \varphi_j^{\tilde{\tau}}(n)$. If there is, then we extend τ to $\tilde{\tau}$, thereby ensuring that φ_j^C cannot be a generic computation of any real that generically reduces to A via φ_i. (This is because neither computation is allowed to make any mistakes, and in particular, generic reductions are not allowed to make any mistakes on any generic oracles, including the entire oracle.)

On the other hand, if there is no such $\tilde{\tau}$, then we may generically compute any D that generically reduces to A via φ_i, and that is generically computable from C via φ_j by an algorithm that halts on a subset of the halting set of φ_i^A, giving only the outputs given by φ_j applied to extensions of τ.

The key thing to notice here is that the domain of φ_i^A is a density-1 set that generically reduces to A. This is witnessed by the algorithm that halts wherever φ_i halts, and that outputs a 1 wherever it halts. (By the definition of generic reduction, for any generic oracle (A) for A, $dom\big(\varphi_i^{(A)}\big) \subseteq dom\big(\varphi_i^A\big)$. Furthermore, we may assume that for every such (A), $dom\big(\varphi_i^{(A)}\big)$ is density-1, because otherwise φ_i is not a generic reduction when applied to A.)

Thus, by the hypothesis on A, $dom\big(\varphi_i^A\big)$ is generically computable, and so contains a density-1 r.e. subset, W. (By Lemma 1.1.11, a density-1 set is generically computable if and only if it contains a density-1 r.e. subset.)

Now, we define ψ so that $\psi(n)\downarrow$ if and only if $n \in W$ and there exists a $\tilde{\tau}$ extending τ such that $\varphi_j^{\tilde{\tau}}(n)\downarrow$. In this case, we let $\psi(n) = \varphi_j^{\tilde{\tau}}(n)$ for the first such $\tilde{\tau}$ that we find.

This is a generic computation of any D that we are concerned with, because the halting set is contained in the halting set of φ_i^A, and the fact that we couldn't diagonalize means that the output that we found from our $\tilde{\tau}$ must be the same output as the output given by φ_i^A.

□

Thus, not only would **a** be half of a minimal pair, but the other half would not even necessarily be quasi-minimal.

As a corollary to Propositions 2.2.15 and 2.2.16, we may deduce the following fact.

Corollary 2.2.17. *If there exist minimal generic degrees, then there exist minimal pairs of generic degrees. In fact, any minimal generic degree is half of a minimal pair in the generic degrees.*

This is simply because the answer to Question 3 must either be "yes," or "no."

After we posed question 2, we mentioned that the only way for Corollary 2.2.17 to be false would be if there were a single minimal generic degree that was below all the other nontrivial generic degrees. In the Turing degrees, it is easy to show that any nontrivial Turing degree has another Turing degree that is incomparable to it, but we do not have a proof that this is true in the generic degrees. It would seem highly unlikely that the generic degrees have an "hourglass" shape, with a nonminimal generic degree that is comparable to all others (such a degree would necessarily be quasi-minimal), but we do not currently have a method of ruling this out.

Question 4. *Does there exist a generic degree $\boldsymbol{a}$ such that for every generic degree $\boldsymbol{b}$, either $\boldsymbol{a} \geq_g \boldsymbol{b}$, or $\boldsymbol{b} \geq_g \boldsymbol{a}$?*

2.3 Nonuniform Generic Reduction

In this section, we will focus on nonuniform time-dependent generic reduction, $\geq_G$. In general, we will refer to this as simply nonuniform generic reduction, since very little can be said about either nonuniform generic reduction, besides that which is also true of the uniform ones, and the only tool that we know of for working with nonuniform generic reduction only works for the time-dependent version:

Proposition 2.3.1. *$A \equiv_G B$ if and only if $\forall C(C \rightarrow_g A \iff C \rightarrow_g B)$.*

This provides us with a full analogue of Observation 1.1.10, and it also suggests that from a philosophical standpoint, $\geq_G$ might be the "correct" transitive analogue of generic computation, since it shows that the $\equiv_G$ degrees are the degrees which precisely measure how difficult it is to generically compute a given real. (Here, when we discuss how difficult a real is to generically compute, we refer specifically to relative generic computation, $\rightarrow_g$, since that is the relation to which we want a transitive analogue.)

Proof. As in the proof of Observation 1.1.10, we prove the following:

$A \geq_G B$ if and only if $\forall C(C \rightarrow_g A \Longrightarrow C \rightarrow_g B)$.

First, assume that $A \geq_G B$.

Then it is easy to show that $C \rightarrow_g A \Longrightarrow C \rightarrow_g B$, simply because if $C \rightarrow_g A$, then C can compute a (single) generic oracle for A, and that generic oracle can be used to generically compute B.

Conversely, assume that $\forall C(C \rightarrow_g A \Longrightarrow C \rightarrow_g B)$.

Fix a time-dependent generic oracle (A) for A.

Then (A), regarded as a set, can be used to generically compute A (so therefore $(A) \rightarrow_g A$). Thus, by assumption, $(A) \rightarrow_g B$.

Thus, any time dependent generic oracle for A can be used to generically compute B, so $A \geq_G B$.

□

Thus, the non-uniform generic degrees can be regarded entirely as filters in the Turing degrees (with a generic degree corresponding to the set of Turing degrees that embed above it), and with $\geq_G$ interpreted as $\supseteq$.

From this point of view, there is a natural question to ask.

Question 5. *Which filters in the Turing degrees can be realized as the filter above some generic degree?*

We also mention that the answer to this question is independent of the definition of generic reduction that we use. The easiest proof of this is by Corollary 2.1.15, which, in particular, shows that this question can also be thought of as a question about generic computability.

The question is somehow more relevant to $\geq_G$ than it is to $\geq_g$ or to $\rightarrow_g$, since a complete answer to Question 5 would also completely characterize the $\equiv_G$-degrees.

Chapter 3

Related Reductions

3.1 Introduction

This chapter will consist almost entirely of work done in collaboration with Damir Dzhafarov. In this chapter, we introduce and discuss a number of notions similar to generic reduction in terms of their format.

Motivation

The individual reducibility notions that are discussed in this chapter are not always intrinsically important, but the overarching program for this chapter is to analyze and understand two general questions.

Question 6. *How is the theory of a notion or reducibility dependent on the specifics of the involved definitions?*

Question 7. *What sorts of strategies can be used to code information in a way so that it can be recovered when the rules for recovering information from an oracle are unusual?*

An understanding of Question 6 should provide us with the ability to shift mindsets between different definitions, and different contexts, and to make accurate predictions about how the change in context will affect the results that can be concluded in that context. This sort of mental agility is useful when proving or disproving almost any analogue of almost any theorem, because it helps us find and focus on the key difference that must be overcome or exploited in the new context.

Question 7 is motivated in large part by reverse mathematics. The most common theorems analyzed in reverse mathematics are theorems of the form "For every set A satisfying certain hypotheses, there is a set B with certain properties." In practice, when one proves that such a theorem is reverse mathematically powerful, the proof involves building A recursively while ensuring that any B satisfying the conditions of the theorem must be computationally powerful. This can be thought of as a fairly obscurely defined rule for extracting information from A, and in that sense, it is important to know how to build A so that the desired information can be recovered from any B.

Definitions

When working with generic reduction, to show that $A \leq_g B$, one insists that the oracle gives outputs on density-1, and that it is correct wherever it gives outputs, and from this, one attempts to produce density-1 many bits of B. In this chapter, we consider altering the definition, both by using notions of largeness other than "density-1," and by asking what happens if we demand that oracles and computations are total, but we only demand that they are correct on a "large" set.

As with generic reduction, we could also work with both uniform and nonuniform reducibilities, but for many of our reducibilities, the nonuniform ones will be trivial, and for many of the rest, the nonuniform ones will be too difficult to work with. We will still mention nonuniform reducibilities, but they will not be the focus of the chapter.

It should also be mentioned that the "largeness" notions being considered do not necessarily need to satisfy the usual hypotheses of a largeness notion. In particular, we do not necessarily demand that the intersection of two large sets is large, nor do we demand that every superset of a large set is large.

We now define all of the reducibilities that will be studied in this chapter, and briefly explain why we use those specific reducibilities.

Definition 3.1.1. *B is coarsely reducible to A, or $A \geq_{cor} B$ if from any oracle for A which is correct on a density-1 set, one can coarsely compute B in the sense of Definition 1.1.12.*

For this definition, the uniform and nonuniform notions of coarse reducibility can both be studied, and we will consider both, although the main result that we will prove is that neither implies, nor is implied by either Turing reducibility or generic reducibility.

Definition 3.1.2. *B is mod-finitely reducible to A, or $A \geq_{mf} B$ if from any oracle for A which is correct on a cofinite set, one can uniformly produce a computation of B which is correct on a cofinite set.*

For this definition, the uniformity hypothesis is essential, because if we use the nonuniform definition, the reduction is equivalent to Turing reduction. We also consider the analogous notion with partial oracles, *cofinite reduction.*

Definition 3.1.3. *B is cofinitely reducible to A, or $A \geq_{cf} B$ if from any partial oracle for A which halts on a cofinite set, one can uniformly produce a partial computation of B which halts on a cofinite set (and always gives correct answers.)*

The uniformity hypothesis is again essential, because nonuniform cofinite reducibility is equivalent to Turing reducibility. We compare both of these to a related notion that is defined in terms of restricting our computation techniques, rather than in terms of altering our oracles and our outputs.

Definition 3.1.4. B is *use-bounded-from-below reducible* to A, or $A \geq_{ubfb} B$ if there is a Turing functional φ such that φ^A is a computation of B, and such that for every n, there is an m_0 such that for all $m \geq m_0$, φ never queries n while computing $\varphi^A(m)$.

The idea behind this reduction is that normally, the use of a computation is thought of in terms of the largest element of the oracle that is queried during the computation, but here, we care about what the smallest queried number is, or in other words, whether the use for the computation is eventually bounded from below. (Again, we remind the reader that in our model of oracle computations, individual bits of the oracle can be queried, rather than initial strings of the oracle. This is classically equivalent, but generalizes better to our setting.)

Morally speaking, a reduction in which every bit of the oracle only needs to be queried for finitely many bits of the output should be closely related to a reduction in which missing finitely much information about the oracle causes us to be able to correctly compute all but finitely many bits of the output, since such a reduction algorithmically captures the essence of each bit of the oracle only being necessary for finitely many bits of the output. However these notions are not equivalent. We show that $(A \geq_{mf} B) \Rightarrow (A \geq_{ubfb} B) \Rightarrow (A \geq_{cf} B) \Rightarrow (A \geq_T B)$, and that none of the implications reverse.

We also define some reductions that are more counterintuitive at first glance.

Definition 3.1.5. B is *mod-recursively reducible* to A, or $A \geq_{mr} B$ if from any oracle for A which is correct on a recursive set, one can uniformly produce a computation of B which is correct on a recursive set.

Here, it is important to understand that when we say that the oracle is correct on a recursive set, we mean that the set on which it is correct is recursive. We do not mean that the set on which it is correct contains a recursive set. This does not produce any particular problems, and is our “largeness” notion that is not closed under supersets. However, it becomes meaningless to attempt to work with this “largeness” notion in terms of partial oracles (since the empty set is recursive, and a computation that never halts is easy to produce.) We show that $(A \geq_{mr} B) \Rightarrow (A \geq_T B)$, and that the implication is strict.

Finally, we discuss our most counterintuitive reduction (not incredibly strange in terms of its definition, but very strange in terms of its behavior.)

Definition 3.1.6. B is *infinite information reducible* to A, or $A \geq_{ii} B$ if from any partial oracle for A which halts on an infinite set, one can uniformly produce a partial computation of B which halts on an infinite set (and always gives correct answers where it halts.)

We show that the usual join of two reals becomes the infimum of their degrees in the ii degrees. We show that 1-reductions embed backwards in the ii degrees. We also show a highly counterintuitive result: that there exist maximal pairs in the ii degrees.

Related Nontransitive Computations

We now present one additional piece of notation, allowing us to more easily discuss the sorts of procedures that are involved in working with the reductions that we have presented thus far, and also giving us more flexibility in discussing relationships between the different degree structures that we work with.

The definition is slightly imprecise for reasons that are somewhat unavoidable. In practice, for specific a and b, the definition will usually be clear, and when it is not clear, we will clarify what we mean.

Definition 3.1.7. *Let a and b be two different notions of reduction, and let A and B be reals.*

Then, $A \rightharpoondown_{a,b} B$ if an a-oracle for A can be used to produce a b-computation of B. In this case, we also write $B \leftharpoondown_{b,a} A$.

If $A \rightharpoondown_{a,b} B$, and $B \rightharpoondown_{b,a} A$, then we write $A \leftrightarrow_{a,b} B$

So, for instance, we have that $\mathcal{R}(A) \geq_g B$ if and only if $A \rightharpoondown_g B$, which is equivalent to $A \rightharpoondown_{T,g} B$. Likewise, $A \geq_g \mathcal{R}(B)$ if and only if $A \rightharpoondown_{g,T} B$. The proof that $X \mapsto \mathcal{R}(X)$ induces an embedding from the Turing degrees to the generic degrees follows from the notion of "transitivity" that is frequently satisfied by such notions of computation.

Definition 3.1.8. Let a, b, and c be reduction notions. Then $\rightharpoondown_{a,b}$ and $\rightharpoondown_{b,c}$ are *transitive with respect to each other* if, for any reals A, B, C, if $A \rightharpoondown_{a,b} B$, and $B \rightharpoondown_{b,c} C$, then $A \rightharpoondown_{a,c} C$.

Lemma 3.1.9. *Assume a, b, c are any of the following reduction notions.*

T, g, cor, cf, mf, mr, ii.

Then $\rightharpoondown_{a,b}$ and $\rightharpoondown_{b,c}$ are transitive with respect to each other.

Proof. All of the above notions have the property that working with a b-oracle for B involves being able to uniformly do something from any b-computation of B. Also, none of the above notions have restrictions on what a Turing machine is allowed to do over the course of a computation.

(Here, for all of the "partial" reduction notions, we use the time-dependent formalizations, which ensures that all of the actual internal work being done during a computation is just work with ordinary Turing machines. It is not necessarily true that time-dependent and time-independent oracles are equivalent when the output is allowed to have some mistakes.)

Thus, to c-compute C from an a-oracle for A, we first use the a-oracle for A to b-compute B, and then use the outputs of that computation as our b-oracle to c-compute C.

□

From this lemma, the proof of Observarion 2.1.12, that $X \mapsto \mathcal{R}(X)$ induces an embedding from the Turing degrees to the generic degrees, can be reduced to the following.

Proof. For any X, $X \leftrightarrow_{T,g} \mathcal{R}(X)$.

Thus, if $A \geq_T B$, then $\mathcal{R}(A) \rightarrow_{g,T} A \rightarrow_{T,T} B \rightarrow_{T,g} \mathcal{R}(B)$, so $\mathcal{R}(A) \geq_g \mathcal{R}(B)$.

Likewise, if $\mathcal{R}(A) \geq_g \mathcal{R}(B)$, then $A \rightarrow_{T,g} \mathcal{R}(A) \rightarrow_{g,g} \mathcal{R}(B) \rightarrow_{g,T} B$, so $A \geq_T B$.

□

Notice now that over the course of proving Proposition 2.1.14, that $\geq_G$ and $\geq_g$ are not equivalent, we also proved that for any X, $X \leftrightarrow_{cf,g} \widetilde{\mathcal{R}}(A)$.

Corollary 3.1.10. *The map $X \mapsto \widetilde{\mathcal{R}}(X)$ induces an embedding of the cofinite degrees into the generic degrees.*

We also mention that, in both directions of our proof of Theorem 2.2.14, that A is hyperarithmetic if and only if there is a density-1 real B such that $B \geq_g \mathcal{R}(\mathcal{A})$, we used the fact that the hyperarithmetic reals are precisely those that can be computed from sufficiently fast growing functions. However, we did *not* establish a strong equivalence between sufficiently fast-growing functions and density-1 generic oracles.

To formalize this, let "fg" be the "sufficiently fast growing" notion of reduction. (So a $\rightarrow_{fg,-}$ computation is a computation that can be done uniformly with any oracle g such that for all n $g(n) \geq f(n)$, and a $\rightarrow_{-,fg}$ f computation is a computation which produces a function g such that for all n $g(n) \geq f(n)$.)

Then, in our proof of Theorem 2.2.14, we showed that for every f, there is a density-1 real B such that $B \rightarrow_{g,fg} f$. (Here, again, we use the time-dependent notion of generic computations.)

We also showed that for any density-1 real B, there is a real f such that for any real A, if $B \rightarrow_{g,T} A$, then $f \rightarrow_{fg,T} A$, but f did not have the property that $f \rightarrow_{fg,g} B$.

Thus, anything that can be done with a sufficiently fast growing function can also be done with a generic oracle for some density-1 set, but the converse is only true for which reals can be Turing computed.

3.2 Coarse Reduction

Recall that B is coarsely reducible to A, or $A \geq_{cor} B$ if from any oracle for A which is correct on a density-1 set, one can coarsely compute B. We present the same definition more rigorously

Definition 3.2.1. B is *coarsely reducible* to A, or $A \geq_{cor} B$ if there exists a Turing functional φ such that for any C, if the set of n such that $C(n) = A(n)$ has density 1, then φ^C is a total function, and the set of n such that $\varphi^C(n) = B(n)$ is density-1.

If the Turing functional is allowed to depend on C, then B is non-uniformly coarsely reducible to A. The two reductions are fairly similar, and all of our proofs function equally well for either reduction.

We show that neither coarse reducibility nor Turing reducibility implies the other.

Proposition 3.2.2. *There exist A and B such that $A \geq_T B$ but $A \not\geq_{cor} B$.*

Proof. Let B to be any real that is not coarsely computable, and let $A = r(B)$. ($r(B)$ is the real where the bits of B are coded sparsely into the powers of 2, from Definition 1.1.5.)

B and $r(B)$ are in the same Turing degree, so $A \geq_T B$.

On the other hand, $A \not\geq_{cor} B$, because 0 is an oracle which agrees with A on a set of density 1, and B cannot be coarsely computed without an oracle. (This proof functions for both uniform and nonuniform coarse reductions.)

□

Proposition 3.2.3. *There exist A and B such that $A \geq_{cor} B$ but $A \not\geq_T B$.*

Proof. Let C be any nonrecursive real. Let $A = 0$. Let $B = r(C)$.

$r(C)$ is not computable, so $A \not\geq_T B$, but $r(C)$ is coarsely computable without even using an oracle, so $A \geq_{cor} B$.

□

We also show that neither coarse reducibility nor generic reducibility implies the other.

Proposition 3.2.4. *There exist A and B such that $A \geq_g B$ but $A \not\geq_{cor} B$. There also exist A and B such that $A \geq_{cor} B$ but $A \not\geq_g B$.*

Proof. For either direction, let A be zero, and let B be the real constructed from the corresponding part of Proposition 1.1.13. □

3.3 Mod-Finite, and Cofinite Reductions

Our goal for this section will be to prove the following implications, and to prove that none of the implications reverse.

$$(A \geq_{mf} B) \Rightarrow (A \geq_{ubfb} B) \Rightarrow (A \geq_{cf} B) \Rightarrow (A \geq_T B) \tag{3.1}$$

Before we do that, we provide more rigorous formalizations of the definitions of the reductions.

Introduction

Definition 3.3.1. B is *mod-finitely reducible* to A, or $A \geq_{mf} B$ if there exists a Turing functional φ such that φ^A is a computation of B, and such that for any C, if the set of n such that $C(n) \neq A(n)$ is finite, then φ^C is a total function, and the set of n such that $\varphi^C(n) \neq B(n)$ is finite.

We mention here that there is an additional hypothesis that might seem somewhat unwarranted, which is that not only should an almost correct oracle give an almost correct computation, but we also demand that the correct oracle must give a correct computation. We use this additional requirement here because it does not affect what it means for B to be reducible to A, but it does provide a definition that is easier to work with, since it makes it easier to diagonalize against specific potential mod-finite reductions.

Lemma 3.3.2. *Definitions 3.1.2 and 3.3.1 are equivalent.*

Proof. We only need to prove that if there exists a mod-finite reduction according to Definition 3.1.2, then there also exists a mod-finite reduction in which an oracle with no mistakes produces an output with no mistakes.

Let A and B be reals, and let φ be a Turing functional. Assume that for every C that is mod-finitely equal to A, φ^C is mod-finitely equal to B.

Let $S = \{n \mid \varphi^A(n) \neq B(n)\}$.

Define ψ where $\psi^X(n) = \begin{cases} \varphi^X(n) & n \notin S \\ B(n) & n \in S \end{cases}$.

S is finite, and thus the values of B on S can be coded recursively, so ψ is recursive. Also, for every C that is mod-finitely equal to A, ψ^C is mod-finitely equal to φ^C, and thus is mod-finitely equal to B. Finally, $\psi^A = B$ by construction.

□

We also reiterate the comment from the introduction, that we only discuss uniform mod-finite reductions, since nonuniform mod-finite reducibility is equivalent to Turing reducibility. (Any oracle for A that disagrees with A on a finite set can be used nonuniformly to recover A, and any computation that is incorrect on a finite set can be used nonuniformly to produce a correct computation.)

Now, we prove that just as cofinite degrees embed in the generic degrees, the mod-finite degrees embed in the coarse degrees.

Corollary 3.3.3. *The map $X \mapsto \widetilde{\mathcal{R}}(X)$ induces an embedding of the mod-finie degrees into the coarse degrees.*

(Recall that $\widetilde{\mathcal{R}}(X)$ is defined by: $n \in \widetilde{\mathcal{R}}(X) \leftrightarrow m \in X$, where 2^m is the largest power of 2 less than n.)

Proof. It is easy to see that, for any A, $A \rightarrow_{mf,cor} \widetilde{\mathcal{R}}(A)$. This is simply because a finite amount of error in A only produces a finite amount of error in the computation of $\widetilde{\mathcal{R}}(A)$.

Conversely, to show that $\widetilde{\mathcal{R}}(A) \rightarrow_{cor,mf} A$, we use a "voting" algorithm.

To recover the mth bit of A from a coarse oracle, R, for $\widetilde{\mathcal{R}}(A)$, we say that $m \in A$ if and only if at least half of the $n \in [2^m, 2^{m+1} - 1]$ are in R.

This algorithm can only make finitely many mistakes, because every time that it makes a mistake, the density of the set on which $R = \widetilde{\mathcal{R}}(A)$ must drop below $\frac{3}{4}$ one additional time. This is because each bit of A is coded into a section of $\widetilde{\mathcal{R}}(A)$ that takes up half of the numbers up to the end of that section, so if half of the coding numbers for the mth bit are wrong, then one quarter of all of the numbers of R (up to the end of that coding section) are wrong.

Thus, we have that, for any A, $A \leftrightarrow_{mf,cor} \widetilde{\mathcal{R}}(A)$, and so the map $X \mapsto \widetilde{\mathcal{R}}(X)$ induces an embedding of the mod-finie degrees into the coarse degrees.

□

Definition 3.3.4. B is *cofinitely reducible* to A, or $A \geq_{cf} B$ if there exists a Turing functional φ such that φ^A is a computation of B, and such that for any partial oracle (A) for A, if the domain of (A) is cofinite, then φ^C is a partial computation of B, and the domain of φ^C is cofinite.

Again, we mention that our insistence that $\varphi^A = B$ is purely a convenience, and that it does not change the definition of the reducibility, because, if some finite amount of information is missing from φ, then that information can be "hard coded" into the algorithm. Also the proof of Proposition 2.1.10 holds in this context, so since we are working with uniform reductions, we may ignore whether or not the oracle is time-dependent.

We may now prove one of our intended implications.

Lemma 3.3.5. *$(A \geq_{cf} B) \Rightarrow (A \geq_T B)$, and $(A \geq_T B) \not\Rightarrow (A \geq_{cf} B)$.*

Proof. The fact that $(A \geq_{cf} B) \Rightarrow (A \geq_T B)$ follows directly from the fact that a cofinite reduction is assumed to be a Turing reduction with additional properties.

For the nonimplication, let A be any real that is not autoreducible, and let $B = \mathcal{R}(A)$. Then, as with the proof of Proposition 2.1.14, A cannot uniformly be recovered from a cofinite subset of its bits, but any cofinite subset of the bits of B can be used to uniformly recover A, and so it is not possible to uniformly generate a cofinite subset of the bits of B from a cofinite subset of the bits of A since that would involve being able to generate all the bits of A from a cofinite subset of the bits of A.

Thus $A \geq_T B$ but $A \not\geq_{cf} B$.

□

Observation 3.3.6. *The Turing degrees embed in the cofinite degrees. Furthermore, the embedding of the Turing degrees into the generic degrees (from Observation 2.1.12) is equal to the composition of this embedding with the embedding of the cofinite degrees into the generic degrees (from Corollary 3.1.10.)*

Proof. The map $X \mapsto \mathcal{R}(X)$ induces an embedding of the Turing degrees into the cofinite degrees.

This is because X can compute $\mathcal{R}(X)$, so in particular $X \to_{T,cf} \mathcal{R}(X)$.

Conversely, any cofinite oracle for $\mathcal{R}(X)$ must include at least one bit from each column, and X can be recovered from those bits, so $\mathcal{R}(X) \to_{cf,T} X$.

By the usual application of Lemma 3.1.9, this proves that $X \mapsto \mathcal{R}(X)$ induces an embedding of the Turing degrees into the cofinite degrees.

To show the second part, we only need to show that for any X, $\widetilde{\mathcal{R}}(\mathcal{R}(X)) \equiv_g \mathcal{R}(X)$.

We accomplish this by liberal use of Lemma 3.1.9.

$$\widetilde{\mathcal{R}}(\mathcal{R}(X)) \to_{g,cf} \mathcal{R}(X) \to_{cf,g} \mathcal{R}(X) \to_{g,T} X \to_{T,cf} \mathcal{R}(X).$$

The first arrow is by Corollary 3.1.10. The second is since any cofinite oracle for a real is also a generic oracle for that real. The third is by Observation 2.1.12. The fourth is by the first part of the proof of this observation.

□

Use-bounded-from-below reductions

Recall the definition of $\geq_{ubfb}$:

Definition 3.3.7. B is *use-bounded-from-below reducible* to A, or $A \geq_{ubfb} B$ if there is a Turing functional φ such that φ^A is a computation of B, and such that for every n, there is an m_0 such that for all $m \geq m_0$, φ never queries n while computing $\varphi^A(m)$.

We reiterate that the idea of this definition is that every bit of the oracle is only queried for finitely many outputs. (In other words, the "use" of $\varphi^A(m)$ is bounded from below by a function that depends on m and that eventually goes to infinity.)

Lemma 3.3.8. $(A \geq_{ubfb} B) \Rightarrow (A \geq_{cf} B)$.

Proof. This implication is fairly easy, since any use-bounded-from-below reduction actually *is* a cofinite reduction:

Assume that $A \geq_{ubfb} B$ via φ. Then φ also provides a cofinite reduction, if it is simply assumed to not halt whenever it queries a bit of its oracle, and the oracle does not give an answer. Then, given any cofinite oracle, (A), for A, choose m_0 large enough so that for all $m \geq m_0$, φ never queries any n that is not in (A) while computing $\varphi^B(m)$.

Then $\varphi^{(A)}(m)$ halts (and is equal to $B(m)$) for any $m \geq m_0$.

Thus, $A \geq_{cf} B$ via φ.

□

Lemma 3.3.9. $(A \geq_{cf} B) \not\Rightarrow (A \geq_{ubfb} B)$.

This is our first involved proof in this chapter, and the technique we will use will be used frequently for our nonimplications. It is similar, but not as complicated, as the method used to prove Theorem 2.2.1

The proof will be comprised of three parts.

In the first part, we describe the intended reduction from A to B. This reduction will be a cofinite reduction, but it will not (a priori) be a *ubfb* reduction.

In the second part, we build A and B so that the intended reduction will work, but simultaneously we diagonalize against every potential *ubfb* reduction. During this part, we do not need to work recursively in anything, but in practice, we will be building a 1-generic with respect to the poset that is implicitly defined in Part 1.

In the third part, we verify that the construction from the second part has the intended properties.

Proof. **Part 1**

For each pair $\langle i, j\rangle$, we will have a single bit $b_{\langle i,j\rangle} \in \{0, 1\}$.

Each of these bits is coded into a single bit of B.

On the other hand, for each value of i, the bits can be coded into A in one of two manners:

Sometimes there is a "master deduction procedure" that, for our fixed value of i, can be used to recover every $b_{\langle i,j\rangle}$, and sometimes there exists a collection of "individual deduction procedures" to recover each of the individual values of $b_{\langle i,j\rangle}$.

For any value of i, if there is no master deduction procedure, then every individual deduction procedure will exist. On the other hand, if there is a master deduction procedure, then every individual deduction procedure that gives an answer must give the same answer as the master deduction procedure, and furthermore all but finitely many of the individual deduction procedures will give answers. Individual deduction procedures are defined in a way so that when they do not give answers, it is never possible to "know" that this is the case.

The purpose of this is to ensure that removing a master deduction procedure from the domain of (A) only removes finitely many bits from the domain of $\varphi^{(A)}$. However, a potential *ubfb* reduction will never know whether or not it needs to query the master deduction.

The deduction procedures are coded as follows.

For each i, we assign two bits, $a_{i,0}, a_{i,1} \in \{0, 1\}$ to the master deduction procedure for i.

For each i, at most one of $a_{i,0}$ and $a_{i,1}$ will be equal to 1.

If $a_{i,0} = 1$, then $\forall j \;\; b_{\langle i,j\rangle} = 0$.

If $a_{i,1} = 1$, then $\forall j \;\; b_{\langle i,j\rangle} = 1$.

Each of these bits is coded into a single bit of A.

Also, for each $\langle i, j\rangle$, we assign countably many bits $a_{\langle i,j\rangle,k,0}$, $a_{\langle i,j\rangle,k,1}$ to individual deduction procedures.

For each $\langle i, j\rangle$, at most one of the $a_{\langle i,j\rangle,k,0}$, $a_{\langle i,j\rangle,k,1}$ will be equal to 1.

If $\exists k \, a_{\langle i,j\rangle,k,0} = 1$, then $b_{\langle i,j\rangle} = 0$.

If $\exists k \, a_{\langle i,j\rangle,k,1} = 1$, then $b_{\langle i,j\rangle} = 1$.

Each of these bits is also coded into a single bit of A.

Then, the idea is that any reduction, while trying to compute $b_{\langle i,j\rangle}$, if it has not yet seen an individual deduction, needs to make a choice. It can guess that there exists a master deduction procedure, and look at the individual deduction of a related b, and thereby risk giving an incorrect answer if there is no master deduction. It can also guess that there is no master deduction procedure, and keep waiting for the individual deduction procedure to halt, and thereby risk not being total if that deduction procedure does not hat. It can also give up, and query the master deduction procedure, but for a fixed value of i, it can only afford to do this for finitely many values of j, or else it will not be a *ubfb* reduction.

Part 2

We build A (and therefore also B) by finite approximation.

At stage s, we have determined some finite segment of A. Choose i_s minimal such that, for every $i \geq i_s$ we have not yet determined any of the values of the $a_{i,0}$, $a_{i,1}$, $a_{\langle i,j\rangle,k,0}$, or $a_{\langle i,j\rangle,k,1}$.

Then, first we ask whether there is any way to extend our definition of A (consistently with ensuring that the requirements from Part 1 are satisfied, and also with ensuring that the restrictions imposed by higher priority strategies are satisfied) in order to make φ_s^A incorrectly compute $b_{\langle i_s,j\rangle}$ for some j.

If there is, then we finitely extend our definition of A to make that happen, and we have successfully prevented B from being $\leq_{ubfb} A$ via φ_s.

If there is no way to make φ^A incorrect, then we ensure that there exists a master computation for i_s, and we initiate a substrategy that will ensure that if $\leq_T A$ via φ_s, then φ_s^A is not a $ubfb$ reduction, while simultaneously ensuring that the intended reduction is a cofinite reduction.

This substrategy fixes some value, $\widetilde{j}$, for j, and it ensures that, unless it decides to choose a new $\widetilde{j}$, the individual deduction for computing $b_{\langle i_s,\widetilde{j}\rangle}$ never gives an output. Then, in the future, if φ_s ever queries a "forbidden element" (defined in the next parahraph) when computing the value of $b_{\langle i_s,\widetilde{j}\rangle}$, then the strategy chooses a new $\widetilde{j}$ for which no individual deduction computing $b_{\langle i_s,\widetilde{j}\rangle}$ has given an output, and allows the previous individual deduction to give the correct output.

Here, the forbidden elements include the two elements of A coding the master deduction for i_s, and they also include, for every $r < s$, if the rth substrategy had been initiated at stage r, if we let $j_{r,s}$ be the value of j that the rth substrategy was restraining at the start of stage s, if that strategy ever acts again, then location $l_{r,s}$ where the individual deduction for $b_{\langle i_r,j_{r,s}\rangle}$ is coded is also a forbidden element. (So notice that at stage s, the locations of the forbidden elements are not all defined, but there are finitely many forbidden elements, and once an element becomes forbidden, it never stops being forbidden, so if φ_s ever queries a forbidden element, at some state of the construction, we will know that it did so, and that that point, the sth substrategy can look for a new $\widetilde{j}$.)

After the strategy has acted, all higher priority substrategies that need to act are allowed to act, and also all individual deductions computing $b_{\langle i,j\rangle}$ for $i, j \leq s$ are made to give outputs unless they are restrained by some strategy, or unless they have already given an output. This ensures that all strategies will act infinitely often, and that all individual deduction procedures will eventually give outputs unless actively restrained by some strategy.

This completes the construction.

Part 3

First, we verify that the intended reduction is a cofinite reduction.

First of all, with the entire oracle for A, B can be computed because for every bit of B, the value of that bit can be deduced either from a master procedure, or from an individual procedure (or frequently both.) Secondly, if the oracle does not halt on finitely many bits of A, then this only prevents the computation from halting on finitely many bits of B. This is because if the oracle for A fails to give the value of A at a place where A has a zero, then this does not affect the computation at all. If it fails to give the value of A at a 1 that is in one of the individual procedures, then this prevents at most one bit of B from being correctly deducible. (Depending on whether or not the oracle includes a master procedure for that bit.) If a 1 is removed from a master procedure, then this theoretically makes it more difficult to deduce the values of B at all of the related bits, however there will still exist individual computations for all but at most one of the bits that are deducible from that master procedure. This is due to the fact that every individual procedure eventually gives an output unless it is specifically restrained by some substrategy, each substrategy only restrains one individual procedure at a time, and each substrategy works on a different value of i (so in particular, two different substrategies will always work on bits that are dependent on different master procedures.)

Thus, the intended reduction can be made to attempt to compute a given bit by simultaneously attempting to query the master deduction bits, and also searching all of the individual deduction bits that are tied to that bit. Thus, removing a finite amount of information from the oracle removes at most a finite number of places where the computation halts. (Recall that this is the paradigm of computation in which oracles and computations are never allowed to be incorrect, but are sometimes allowed to not give outputs.)

Thus, $A \geq_{cf} B$.

Now, we verify that $A \not\geq_{ubfb} B$.

For this, we must show that for every s, either φ_s^A is not a computation of B, or φ_s^A is not a use-bounded-from-below computation.

So, fix some s, and let i_s be the i that was chosen at stage s. If, at stage s, A had been able to be extended to make φ_s produce a mistake, then this was done, and so $\varphi_s^A \neq B$.

On the other hand, if not, then one of two things would have happened during the construction. One option is that the corresponding substrategy acted infinitely often, in which case there were infinitely many different $b_{\langle i_s, \widetilde{j} \rangle}$ such that φ_s queried a forbidden element. Since there are finitely many elements that are forbidden to φ_s, by the pigeonhole principle this means that there must be one bit of A that is queried by φ_s during the computations of infinitely many different bits of B, so φ_s^A is not *ubfb*.

Alternatively, it is possible that the corresponding substrategy acted only finitely often. In this case, there was one fixed $\widetilde{j}$ such that φ_s never queried the master deduction procedure, or any of the other forbidden bits, when computing the value of $b_{\langle i_s, \widetilde{j} \rangle}$, and such that the individual deduction procedure for $b_{\langle i_s, \widetilde{j} \rangle}$ never gave any outputs. Let n be the location in B where $b_{\langle i_s, \widetilde{j} \rangle}$ is coded. (So $n \in B \Leftrightarrow b_{\langle i_s, \widetilde{j} \rangle} = 1$.) In this case, we claim that $\varphi_s^A(n)$ did not halt.

The reason for this is that otherwise, at the stage when $\varphi_s^A(n)$ halted, the fraction of A that φ_s had looked at was consistent both with $n \in A$, and $n \notin A$ (because the master deduction was not queried, and the individual deduction never gave any output.) It was also consistent with the restrictions imposed at the beginning of stage s (because none of the other forbidden elements were queried, so the fraction of A that had been looked at did not include any of the outputs of any of the individual deductions that had been restrained at the beginning of stage s.)

Thus, if that were enough to make φ_s halt, then at stage s, it would have been possible to extend our definition of A in order to make $\varphi_s^A(n)$ give an incorrect answer, by simply not including a master computation for column i_s, and by adding an individual computation for $b_{\langle i_s, \tilde{j} \rangle}$ that disagrees with the value of $\varphi_s^A(n)$, and defining all other queried bits exactly the way that they had been defined in A, and defining the forbidden bits to be zero. (This individual computation can be added with a k value that is larger than any of the k values that were looked at by φ_s while it was computing $B(n)$.)

Since, at stage s, none of these values had yet been decided for A, and since this is consistent with every value of A that had been queried by φ_s, and since this does not conflict with any of the restraints imposed by higher priority strategies at the beginning of stage s, this this means that at stage s, the partial definition of A could have been extended to make φ_s^A incorrectly compute B, contradicting the assumption on s.

□

In stark contrast to the incomparability between generic reductions and coarse reductions, mod-finite reductions are somewhat stronger than cofinite reductions, and in fact can be shown to be strictly stronger than *ubfb* reductions.

Lemma 3.3.10. $(A \geq_{mf} B) \Rightarrow (A \geq_{ubfb} B)$.

Proof. Assume that $A \geq_{mf} B$ via φ.

Thus, $\varphi^A = B$ and if $\widetilde{A}$ differs from A on a finite set, then $\varphi^{\tilde{A}}$ computes a set which differs from B on a finite set.

To *ubfb* compute B from A, we define ψ where $\psi^X(n)$ is computed as follows.

Let $X_1, ..., X_{2^n}$ be the reals which agree with X after the first n bits of X. Then, the first thing that ψ does is compute the values of $\varphi^{X_i}(n)$ for each $i \leq 2^n$. It does this without querying any of the first n bits of X, but querying any of the larger bits of X whenever φ would want it to.

After all of the $\varphi^{X_i}(n)$ have halted, ψ starts to, one at a time, query the largest bits of X that it had previously not been allowed to query, until it rules out all but one of the possible outputs, and then it gives that output.

Then, first of all, ψ^A is total, because $\varphi^{A_i}(n)$ will halt for every i, because $\varphi^{\tilde{A}}$ is a total function for any $\widetilde{A}$ that differs from A on a finite set, and each of the A_i differs from A on at most a finite set. At this point, theoretically if ψ were to query all of the first n bits of A, then it would definitely narrow down the possible outputs to one output, so $\psi^A(n)$ will eventually halt.

Also, ψ^A is a computation of B, because the one output that ψ^A cannot rule out is $\varphi^A(n)$, which is equal to $B(n)$, so when it halts, it must give the correct answer.

Finally, ψ^A is a *ubfb* reduction. This is because φ is a mod-finite reduction, so for every a, there is an b such that if $\widetilde{A}$ is equal to A on every number larger than a, then $\varphi^{\tilde{A}}$ is equal to B on every number larger than b. Thus, when $n \geq b$, ψ will not need to query any of the first a many bits of A, since all of those variations of A will give the same outputs on n. Thus, each a will only be queried when computing $\varphi^{\tilde{A}}(n)$ for n less than the corresponding b.

□

Lemma 3.3.11. $(A \geq_{ubfb} B) \not\Rightarrow (A \geq_{mf} B)$.

Once again, the proof is in three parts.

First, we establish the intended reduction by which $A \geq_{ubfb} B$.

Secondly, we construct A and B to ensure that $A \not\geq_{mf} B$.

Finally, we verify that our construction has succeeded.

Proof. **Part 1**

In this construction, each column of A will be used to code a single bit of B. A will have exactly one element in each column, and the parity of that element will tell us whether or not the corresponding number is in B.

More precisely, for each n, there is exactly one m such that $\langle n, m\rangle \in A$. Then, $n \in B \Leftrightarrow \langle n, m\rangle \in A$ for some odd value of m.

This reduction is clearly a *ubfb* reduction, since each bit of A needs to be queried to compute at most one bit of B. However, A is not necessarily mod-finitely above B, since changing a single bit of A can cause the reduction to stop being total. If the reduction can halt without finding any bits in the nth column of A, then the reduction can also be tricked into giving incorrect answers even on the correct oracle (for example, if the element of A in the nth column is very large.)

The remainder of the proof consists of verifying that A, and B can indeed be constructed in this manner so that $A \not\geq_{mf} B$.

We build A (and therefore B) by finite approximation.

Let σ_s be our approximation to A at the beginning of stage s. Then, the first thing we do at stage s is ask whether there is an extension of σ_s, following the rules of the construction, that would cause φ_s^A to incorrectly compute B. If there is, then we make that extension, and we move on to the next stage. Otherwise, let n_s be the smallest value of n such that we have not yet determined whether or not any $\langle n, m\rangle \in A$. Then we put $\langle n_s, 0\rangle \in A$ (and therefore $n \notin B$) and move on to the next stage.

A is then the limit of the σ_s.

Then, we claim that for each s, either $\varphi_s^A \neq B$, or there is some $\widetilde{A}$ that is equal to A on a cofinite set such that $\varphi_s^{\tilde{A}}$ is not total.

If φ_s^A is not total, then we are finished.

Otherwise, we ask whether, at stage s, there was any way to extend σ_s in order to cause φ_s to make a mistake. If yes, then we must have done that, and therefore $\varphi_s^A \neq B$.

If not, then let $\widetilde{A} = A \backslash \{\langle n_s, 0\rangle\}$. Then we claim that $\varphi_s^{\tilde{A}}(n_s)$ diverges.

The proof of this is simply that if $\varphi_s^{\tilde{A}}(n_s)$ converges and gives some answer, then our σ_s could have been extended to have all of the values of $\widetilde{A}$ that were queried before $\varphi_s^{\tilde{A}}(n_s)$ halted, and then to have $\sigma_{s+1}(\langle n_s, m\rangle) = 1$ for some sufficiently large m of the correct parity to make φ_s^A give an incorrect answer on n_s.

□

From the previous five results, we can conclude the primary theorem of the section.

Theorem 3.3.12.
$(A \geq_{mf} B) \Rightarrow (A \geq_{ubfb} B) \Rightarrow (A \geq_{cf} B) \Rightarrow (A \geq_T B)$ *and none of the implications reverse.*

3.4 Mod-Recursive Reductions

Per usual, we begin with a more formal definition of mod-recursive reductions.

Definition 3.4.1. A is *mod-recursively reducible* to B, or $B \geq_{mr} A$ if there exists a Turing functional φ such that φ^A is a computation of B, and such that for any C, if the set of n such that $C(n) \neq A(n)$ is recursive, then φ^C is a total function, and the set of n such that $\varphi^C(n) \neq B(n)$ is recursive.

Again, the requirement that $\varphi^A = B$ does affect the definition of a mod-recursive reduction, but it does not affect the definition of what is mod-recursively reducible to what. We also mention that since $\mathbb{N}$ is a recursive set, it is very important to the definition that the "mistakes" must actually be a recursive set, and not simply contained in a recursive set.

Lemma 3.4.2. *Definitions 3.1.5 and 3.4.1 are equivalent.*

Proof. The proof is almost identical to the proof of Lemma 3.3.2. The only real difference is that the set of places where φ^A makes a mistake is not finite, and so we cannot directly code the values of B at those locations in to ψ. The fix is relatively easy though.

Let A and B be reals, and let φ be a Turing functional. Assume that for every C that is mod-recursively equal to A, φ^C is mod-recursively equal to B.

Let $S = \{n \mid \varphi^A(n) \neq B(n)\}$.

Define ψ where $\psi^X(n) = \begin{cases} \varphi^X(n) & n \notin S \\ 1 - \varphi^X(n) & n \in S \end{cases}$.

S is recursive, and thus ψ is recursive, since it can compute S and emulate φ simultaneously. Also, for every C that is mod-recursively equal to A, ψ^C is mod-recursively equal to φ^C, and thus is mod-recursively equal to B. Finally, $\psi^A = B$ by construction.

□

We mention that this construction would have worked perfectly well for Lemma 3.3.2, but it would not have worked for the proof that the first and second notions of cofinite reducibility were equivalent.

Again, we do not discuss nonuniform mod-recursive reductions, because they are equivalent to Turing reductions. This time, however, we also do not discuss reductions in which the computations are allowed to have a recursive set on which they do not halt, because that reduction would be degenerate, since computations would not need to halt anywhere.

Also, again, mod-recursive reducibility clearly implies Turing reducibility, and our main result in this section will be that it is not equivalent to Turing reducibility.

Proposition 3.4.3. *There exist A and B such that $A \geq_T B$ but $A \not\geq_{mr} B$.*

Proof. The construction is fairly straightforward. B will be the real given by $n \in B$ if and only if both $2n \in A$, and $2n+1 \in A$.

Then, if we use the intended reduction, it will have the property that changing A on all of the odd numbers will change the output on a set that has no reason to be recursive, since whether or not the output changes will depend on whether or not the corresponding even number is in A. As long as the even numbers in A are not a recursive set, this will result in a nonrecursive change in the output.

The remainder of the proof will be constructing A so that no other φ yields a mod-recursive reduction.

We construct A (and therefore B) by finite approximation.

At stage s, let σ_s be our current approximation to A. Let $k_s = |\sigma_s|$. First, as usual, we ask whether there is an extension of σ_s that ensures that $\varphi_s^A \neq B$. If there is such an extension, we make that extension, otherwise we do nothing.

After that, we extend our approximation to ensure that φ_s (without oracle) does not correctly compute the even bits of A in the usual manner in which one diagonalizes against sets being computable. (Fix some n for which we have not yet determined whether 2^n is in A. If $\varphi_s(2n) = 0$, then $2n \in A$. Otherwise, $2n \notin A$. Then extend the approximation so that it is defined on an initial segment of the natural numbers.)

Now, let $\widetilde{A}$ be defined by

$$2n \in \widetilde{A} \Leftrightarrow 2n \in A, \text{and } 2n+1 \in \widetilde{A} \Leftrightarrow 2n+1 \notin A.$$

Also, let A_s be defined by

$$n \in A_s \Leftrightarrow (n < k_s \wedge n \in A) \vee (n \geq k_s \wedge n \in \widetilde{A}).$$

Then we claim that at the end of the construction, for every s, if both φ_s^A, and $\varphi_s^{A_s}$ are total, then either φ_s^A is not a computation of B, or, $\{n \mid \varphi_s^{A_s}(n) \neq B(n)\}$ is not recursive.

Assume φ_s^A, and $\varphi_s^{A_s}$ are total. Then, at stage s, if we could have caused φ_s^A to give an incorrect output, then we would have done that. Thus, we may assume that for any $X \succ \sigma_s$, if $\varphi_s^X(n)\downarrow$, then $\varphi_s^X(n) = X(2n) \cdot X(2n+1)$. (Otherwise, σ_s could have been extended to the portion of X that was used in the computation of $\varphi_s^X(n)$, and this would have caused it to make a mistake.)

Thus, we have that for every n, $\varphi_s^{A_s}(n) = A_s(2n) \cdot A_s(2n+1)$. So, in particular, for $n \geq \frac{k_s}{2}$, we have that $\varphi_s^{A_s}(n) = B(n)$ if and only if $2n \notin A$. Thus, since $\{2n \,|\, 2n \in A\}$ is not a recursive set, $\{n \,|\, \varphi_s^{A_s}(n) \neq B(n)\}$ is not recursive.

□

3.5 Infinite Information Reductions

Our final reduction that we consider is also our most foreign one.

Definition 3.5.1. B is *infinite information reducible* to A, or $A \geq_{ii} B$ if there exists a Turing functional φ such that for any partial oracle (A) for A, if the domain of (A) is infinite, then φ^C is a partial computation of B, and the domain of φ^C is infinite.

Here, we cannot assume that φ^A is a computation of B, because that would dramatically change the nature of the reducibility. Also, only the notion with partial oracles is worth considering, because it is too easy to produce a computation that is correct infinitely often if the computation is also allowed to be incorrect infinitely often (In fact, there would be only one degree under that reduction.)

We begin by showing a single fact that helps illustrate the counterintuitive nature of this reduction.

Observation 3.5.2. *Let A and B be reals. Then the infinite information degree of $A \oplus B$ is the infimum of the ii degrees of A and B.*

Before we begin this proof, we mention that when we write $A \oplus B$, we mean the usual join of A and B (produced by coding A into the even bits of $A \oplus B$, and B into the odd bits of $A \oplus B$). We recognize that this is not the join of the degrees in our current context, but we will still use this notation, since, strictly speaking, $A \oplus B$ is defined as a function on reals, not on Turing degrees.

Proof. The easier part is showing that $A \geq_{ii} A \oplus B$, and likewise that $B \geq_{ii} A \oplus B$.

Intuitively, it is easy to use infinitely many bits of A to correctly produce infinitely many bits of $A \oplus B$. More formally, $A \geq_{ii} A \oplus B$ via the algorithm φ where if (X) is a partial oracle, then $\varphi^{(X)}(n)$ halts only if n is even, and if $(X)(\frac{n}{2})$ halts. In this case, $\varphi^{(X)}(n) = (X)(\frac{n}{2})$. Similarly, $B \geq_{ii} A \oplus B$.

The slightly more difficult part is showing that if $A \geq_{ii} C$, and $B \geq_{ii} C$, then $A \oplus B \geq_{ii} C$.

To show this, assume that φ_0 and φ_1 are Turing functionals such that $A \geq_{ii} C$ via φ_0, and $B \geq_{ii} C$ via φ_1. Then, we can define ψ where for any partial oracle (X) for X, $\psi^{(X)}$ looks at the even bits of X that are in (X), and applies φ_0 to those bits, and it looks at the odd bits of X and applies φ_1 to those bits, and then it outputs anything that either of its two partial functions would output.

Then, $A \oplus B \geq_{ii} C$ via ψ, because any infinite partial oracle for $A \oplus B$ must halt on either infinitely many bits of A or infinitely many bits of B, and thus either the φ_0 part of the φ_1 part of ψ must give infinitely many outputs. Furthermore, for any partial oracle $(A \oplus B)$ for $A \oplus B$, $\psi^{(A \oplus B)}$ does not give any incorrect outputs about C, because φ_0 does not give any incorrect outputs about C with any partial oracle for A, and likewise the φ_1 part does not give any incorrect outputs.

□

At this point, it becomes relevant to show that the *ii* degrees are nontrivial, to ensure that Observation 3.5.2 has actual content.

Observation 3.5.3. *If B is either 1-random relative to A, or 1-generic relative to A, then $A \not\geq_{ii} B$.*

Proof. If we assume, for a contradiction, that $A \geq_{ii} B$ via φ. Then in particular, φ^A must be a partial computation of B that halts on an infinite set and that only gives correct outputs when it halts (Since A is a partial oracle for A with infinite domain). So it remains to show that if B is 1-random, or 1-generic relative to A, then A is not able to correctly compute infinitely many bits of B without making any mistakes.

This is a standard result of classical recursion theory, but we prove it here for completeness:

If φ^A is a partial computation of B which halts for infinitely many n, then B is not 1-random relative to A, since the set of all reals which agree with φ^A is a $\Pi^0_1(A)$ null class. (For each m, the set of reals which agree with φ^A on the first m bits of the domain of φ^A has measure 2^{-m}.)

Likewise, if φ^A is a partial computation of B which halts for infinitely many n, then B is not 1-generic relative to A, because while B was being constructed, it would have always been able to be extended in a way so that it disagreed with φ^A on at least one of φ^A's outputs.

□

From these two results, we can see that *ii* reducibility neither implies nor is implied by Turing reducibility.

Corollary 3.5.4. *There exist A and B such that $A \geq_{ii} B$, $A \not\geq_T B$, $B \not\geq_{ii} A$, and $B \geq_T A$.*

Proof. Let X_1 and X_2 be a pair of mutually 1-random, or mutually 1-generic reals.

Let $A = X_1$, and let $B = X_1 \oplus X_2$. Then $A \geq_{ii} B$ and $B \geq_T A$ because $B = A \oplus C$ for some C.

Also, $A \not\geq_T B$ because $A \not\geq_T X_2$ because X_2 is random (resp. generic) relative to X_1. Likewise, X_1 is random (resp. generic) relative to X_2, so $X_2 \not\geq_{ii} A$ by Observation 3.5.3, and therefore, $B \not\geq_{ii} A$ by Observation 3.5.2, because $B = C \oplus X_2$ for some C.

□

We also generalize Observation 3.5.2 to better illustrate the reason that joins embed upside down in the ii degrees.

Observation 3.5.5. *Let A and B be reals. Then if $A \geq_1 B$, then $B \geq_{ii} A$.*

Proof. We first remind the reader of the definition of $\geq_1$:

$A \geq_1 B$ if and only if there exists a recursive injective function $f : \mathbb{N} \to \mathbb{N}$ such that $n \in B \Leftrightarrow f(n) \in A$.

The idea behind this definition is that B can be computed from A in the (in some sense) easiest possible manner: all of B is embedded in A in a recursive manner, so to compute a single bit of B, you query the corresponding bit of A, and then give that as your output.

However, this can immediately be used to produce an ii reduction from B to A.

$n \in B \Leftrightarrow f(n) \in A$, and so we may define φ where for any partial oracle (X), for any n such that (X) gives an output on n, $\varphi^{(X)}$ gives the same output on $f(n)$. Then, if (B) is a partial oracle for B with infinite domain, then $\varphi^{(B)}$ halts on the image of the domain of (B) under f, which is an infinite set, because f is injective. Furthermore, $\varphi^{(B)}$ correctly computes A wherever it halts because $n \in B \Leftrightarrow f(n) \in A$.

□

We also mention that there certainly exist more intricate ii reductions than those that have been mentioned so far, and in particular, that ii reductions can be built in a way that sometimes requires very many bits of the oracle to produce a single bit of the output.

Proposition 3.5.6. *There exist reals A and B such that $A \geq_{ii} B$ via φ, but such that for any n, there is a finite partial oracle σ, for A such that $\#(dom(\sigma)) = n$, such that φ^σ does not halt anywhere.*

The basic idea of this is that some bits of B can be coded in a way that require a large number of bits of A to recover those bits of B.

Proof. Recursively choose one natural number n_S for each finite set S of natural numbers such that the cardinality of S is equal to the smallest element of S.

Then let $n_S \in B$ if and only if $\#(S \cap A)$ is odd.

Then, first of all, the implicitly defined functional φ is definitely an ii reduction. This is because, once any oracle (A), for A, gives any output on any bit of A, say the nth bit of A, then at that point, any n additional bits of A will be sufficient to compute an additional bit of B.

On the other hand, there is no fixed bound on the number of bits of A required to compute a bit of B: for any n, if (A) is a finite partial oracle for A which halts on n bits, and whose smallest bit is larger than n, then (A) is not sufficient to deduce any bits of B!

□

This proof is somewhat unsatisfying, because any partial oracle (A), for A, will have the property that there is some finite collection of bits in (A) such that knowing just those bits is sufficient to reach the point where from then on, any additional bit of A will produce at least one additional bit of B. This can be easily avoided by replacing A with $A \oplus A$, which would have the property that for almost any partial oracle, there are numerous bits that can be added to the oracle without increasing the domain of the computation.

This new reduction still feels, in some sense, trivial, and, indeed, over the course of proving Theorem 3.5.7, which says that there exist maximal pairs for generic reduction, we will show that many *ii* reductions share certain properties with these types of trivial reductions. However, we will sometimes require hyperarithmetic techniques to reduce to the trivial case.

The remainder of this section will be devoted to establishing some techniques and notations for proving Theorem 3.5.7.

Theorem 3.5.7. *There exist maximal pairs in the ii-degrees.*

In proving this theorem, we will first prove the following result.

Lemma 3.5.8. *Let A and C be reals such that $C \geq_{ii} A$. Then, either $\mathcal{O} \rightarrow_{T,ii} A$, or $A \oplus \mathcal{O} \rightarrow_{T,ii} C$.*

(Here, $\mathcal{O}$ is Kleene's $\mathcal{O}$, defined in Appendix D.)

From this lemma we will immediately be able to conclude a proposition that implies our theorem.

Proposition 3.5.9. *Let A and B be reals such that $A \oplus \mathcal{O} \not\rightarrow_{T,ii} B$, and $B \oplus \mathcal{O} \not\rightarrow_{T,ii} A$. Then there does not exist any real C such that $C \geq_{ii} A$, and $C \geq_{ii} B$.*

Proof. Assuming Lemma 3.5.8, Proposition 3.5.9 follows immediately, since, if there existed a C above both A and B, then either $\mathcal{O} \rightarrow_{T,ii} A$, which would contradict the fact that $B \oplus \mathcal{O} \not\rightarrow_{T,ii} A$, or $A \oplus \mathcal{O} \rightarrow_{T,ii} C$.

But then we would have that $A \oplus \mathcal{O} \rightarrow_{T,ii} B$, by Lemma 3.1.9, since, by assumption, $C \geq_{ii} B$.

□

We now devote our attention to proving Lemma 3.5.8. We will make extensive use of the notations and techniques of [13], and anyone unfamiliar with the language and techniques involved in working with recursive ordinals should look to Appendix D. We also provide a notation for 1-extensions of partial oracles, since they are very relevant for the sorts of finesse that are required for our construction.

Definition 3.5.10. Let σ be a finite partial oracle. (In other words, σ is a partial oracle for some real X, and $\mathrm{dom}(\sigma)$ is finite.)

Then, τ is a *1-extension* of σ, written $\tau \succ_1 \sigma$, if τ is an extension of σ that is defined on exactly one more element. (In other words, if $\#\mathrm{dom}(\tau) = \#\mathrm{dom}(\sigma) + 1$, and if $\tau \restriction \mathrm{dom}(\sigma) = \sigma$.)

To help motivate the upcoming definitions and techniques, we first engage in a mental exercise.

Assume that $C \geq_{ii} A$ via φ. Assume further that σ is a finite partial oracle for C that has the property that for any $\tau \succ_1 \sigma$, if τ is also a partial oracle for C, then $\#\mathrm{dom}(\varphi^\tau) > \#\mathrm{dom}(\varphi^\sigma)$.

Consider the set of all $\tau \succ_1 \sigma$ such that φ^τ does not incorrectly compute any bits of A. (Here, we do not assume that τ is a partial oracle for C. Indeed, every τ that is a partial oracle for C is in this set.) Let X be the set of numbers that are in the domain of φ^τ for one of these τ but that are not in the domain of φ^σ. Then, one of the following things happens.

1. If there exist infinitely many $\tau \succ_1 \sigma$ such that φ^τ incorrectly computes any bits of A, then $A \rightarrow_{T,ii} C$.

   This is because A can search for 1-extensions of σ which give false outputs. Whenever it finds such a 1-extension, it knows that the additional bit that is in $\mathrm{dom}(\varphi^\tau)\setminus\mathrm{dom}(\varphi^\sigma)$ must be incorrect. Thus, it can figure out infinitely many of the bits of C, by ruling out infinitely many incorrect 1-extensions of σ.

2. If there exist finitely many $\tau \succ_1 \sigma$ such that φ^τ incorrectly computes any bits of A, and if X is infinite, then $0 \geq_{ii} A$.

   This is because we may compute infinitely many bits of A by a computation which knows σ, knows all of the τ which give incorrect outputs, and which then gives any output given by φ^τ for any $\tau \succ_1 \sigma$ other than the finitely many "bad" τ's.

3. If there exist finitely many $\tau \succ_1 \sigma$ such that φ^τ incorrectly computes any bits of A, and if X is finite, then, by the pigeonhole principle, there must exist some $n \in X$ such that for infinitely many $\tau \succ_1 \sigma$, $\varphi^\tau(n)$ halts, and is equal to $A(n)$. (Note that by assumption on σ, if $\tau \succ_1 \sigma$ is a partial oracle for C, then $\mathrm{dom}(\varphi^\tau) \supsetneq \mathrm{dom}(\varphi^\sigma)$.)

   In this case, that output must be equal to $A(n)$, since otherwise, we would be in Case (1).

In Case (3), we would like to somehow say that σ already "knew" the value of $A(n)$, and we now define α-deduction to allow us to formalize this idea.

Definition 3.5.11. Let α be an ordinal, σ a finite partial oracle, n an integer, and $i \in \{0,1\}$.

We define the statement *"σ α-deduces that $\varphi(n) = i$"* by induction on α.

σ 0-deduces that $\varphi(n) = i$ if and only if $\varphi^\sigma(n) = i$.

If $\alpha > 0$, then *σ α-deduces that $\varphi(n) = i$* if and only if either $\varphi^\sigma(n) = i$, or there exist infinitely many $\tau \succ_1 \sigma$ such that for each such τ, there is a $\beta_\tau < \alpha$ such that τ β-deduces that $\varphi(n) = i$.

Definition 3.5.12. *We say that "σ deduces that $\varphi(n) = i$" if there exists some α such that σ α-deduces that $\varphi(n) = i$.*

Lemma 3.5.13. *If σ deduces that $\varphi(n) = i$ then there exists some $\alpha < \omega_1^{CK}$ such that σ α-deduces that $\varphi(n) = i$.*

Proof. Fix φ, n, and i.

We first show that the set of finite partial oracles, σ, such that σ deduces that $\varphi(n) = i$ can be defined by an arithmetically definable monotonic closure operator. (Defined in Appendix D.)

Then, since all such closure operators reach their limit at a stage before ω_1^{CK}, we will be able to find an $\alpha < \omega_1^{CK}$ such that for all σ, if σ deduces that $\varphi(n) = i$, then there exists a $\beta \leq \alpha$ such that σ β-deduces that $\varphi(n) = i$.

Let X be a set of finite partial oracles.

Let $\Gamma(X)$ be the set of all finite partial oracles, σ such that any of the following holds.

- $\sigma \in X$
- $\varphi^{\sigma}(n) = i$
- There exist infinitely many $\tau \in X$ such that $\tau \succ_1 \sigma$.

Then, $\Gamma(X)$ is arithmetic in X. (In fact, it is Π^0_2 in X.) Γ is also clearly monotonic.

Furthermore, Γ_α (the αth iteration of Γ on the empty set) is equal to the set of σ such that σ α-deduces that $\varphi(n) = i$.

By Lemma D.0.41, there is an $\alpha < \omega_1^{CK}$ such that $\Gamma_\alpha = \Gamma_{\alpha+1}$.

Then, for that value of α, we have that, for all σ, if σ deduces that $\varphi(n) = i$, then σ α-deduces that $\varphi(n) = i$.

□

Lemma 3.5.14. *If $\alpha < \omega_1^{CK}$, then the statement "σ α-deduces that $\varphi(n) = i$" is uniformly recursive in σ, φ, n, i, and H_a, where a is any ordinal notation for $2 \cdot \alpha + 2$.*

(Ordinal notations, and H_a are defined in Appendix D.)

Proof. The proof is by induction on α.

If $\alpha = 0$, then to determine whether or not σ α-deduces that $\varphi(n) = i$, we simply ask whether $\varphi^{\sigma}(n)$ halts and is equal to i. This is recursive in $0'$, and so also in $0''$.

If α is a successor, then let $\alpha = \beta + 1$.

Then to determine whether or not σ α-deduces that $\varphi(n) = i$, we simply ask whether, $\forall k \exists l \exists \tau$, $(l > k), (\tau \succ_1 \sigma), (l \in \mathrm{dom}(\tau))$, and τ β-deduces that $\varphi(n) = i$.

By induction, whether or not τ β-deduces that $\varphi(n) = i$ is uniformly recursive in H_b, where $|b| = 2 \cdot \beta + 2$. Therefore, this is uniformly recursive in $H_b'' = H_a$ for some a with $|a| = |b| + 2 = 2 \cdot \beta + 4 = 2 \cdot \alpha + 2$.

If α is a limit, then let $u = 3 \cdot 5^e$, and assume $|u| = \alpha$.

Then to determine whether or not σ α-deduces that $\varphi(n) = i$, we ask whether or not it is true that for every k, there is a v that is enumerated by φ_e such that there are k many different $\tau \succ_1 \sigma$ such that τ $|v|$-deduces that $\varphi(n) = i$.

For each v enumerated by φ_e, $|v| < \alpha$, and so $2|v| + 2 < \alpha$ (since α is a limit ordinal). Therefore, H_u can uniformly compute whether or not τ $|v|$-deduces that $\varphi(n) = i$ (since it can uniformly compute all the H-sets required to compute each of those facts), and so $H''_u = H_a$ can compute whether or not the number of τ that $|v|$-deduce that $\varphi(n) = i$ goes to infinity as $|v|$ approaches $|u|$.

□

Lemma 3.5.15. *Let A and C be reals such that $C \geq_{ii} A$ via φ, and assume that for every recursive ordinal and $0^{(\alpha)} \not\twoheadrightarrow_{T,ii} C$. Also, assume that σ is a finite partial oracle for C.*

Then, if σ deduces that $\varphi(n) = i$, then $A(n) = i$.

Note that this assumption on C is implied by $A \oplus \mathcal{O} \not\twoheadrightarrow_{T,ii} C$, since, for every recursive ordinal α, $\mathcal{O} \geq_T 0^{(\alpha)}$. (Here, $0^{(\alpha)}$ is the αth jump of 0, defined in Appendix D.)

Proof. We prove by induction on α that if σ is a finite partial oracle for C, and σ α-deduces that $\varphi(n) = i$, then $A(n) = i$.

If $\alpha = 0$, then clearly if σ α-deduces that $\varphi(n) = i$, and σ is a partial oracle for C, then by definition of 0-deduction, $\varphi^\sigma(n) = i$, and so $A(n) = i$, because $C \geq_{ii} A$ via φ, and so no partial oracle for C is allowed to cause φ to make any mistakes.

Now, let $\alpha > 0$, and assume by induction that for all $\beta < \alpha$, if σ is a finite partial oracle for C, and σ β-deduces that $\varphi(n) = i$, then $A(n) = i$.

Assume that σ is a finite partial oracle for C, and that σ α-deduces that $\varphi(n) = i$.

Then, by definition of α-deduction, we know that there exist infinitely many $\tau \succ_1 \sigma$ such that for some $\beta_\tau < \sigma$, τ β_τ-deduces that $\varphi(n) = i$.

If any of those τ are finite partial oracles for C, then by induction, we can conclude that $A(n) = i$.

Otherwise, none of those τ are finite partial oracles for C. Thus, since σ was a finite partial oracle for C, we know that, for every one of those τ, $\tau(m_\tau) \neq C(m_\tau)$, where m_τ is the single number in the domain of τ that was not in the domain of σ.

By Lemma 3.5.14, we can uniformly compute infinitely many such τ using H_a, where a is a notation for the ordinal $2 \cdot \alpha + 2$. Thus, if none of those τ are finite partial oracles for C, then H_a can compute infinitely many bits of C. It does this by finding those τ, and knowing that the single new bit is incorrect for C.

□

We are now ready to prove Lemma 3.5.8. The proof will be roughly analogous to the mental exercise that was presented after the statement of Lemma 3.5.8, except that the problematic Case (3) will no longer be present.

Proof. Assume that $C \geq_{ii} A$ via φ, and assume that $\mathcal{O} \not\twoheadrightarrow_{T,ii} A$, and $A \oplus \mathcal{O} \not\twoheadrightarrow_{T,ii} C$. For each partial oracle σ, let f_σ be the partial function where $f_\sigma(n) = i$ if and only if σ deduces that $\varphi(n) = i$.

Let σ be a finite partial oracle for C that has the property that for any $\tau \succ_1 \sigma$, if τ is also a partial oracle for C, then $\#\text{dom}(f_\tau) > \#\text{dom}(f_\sigma)$.

Then, first of all, we show that there exists such a σ.

The empty oracle does not deduce infinitely many bits of A, because otherwise $\mathcal{O}$ can compute infinitely many bits of A. This is because $\mathcal{O}$ can uniformly compute all of the H-sets, and thus, by Lemma 3.5.14, can uniformly compute all of the facts that the empty oracle can deduce. (By Lemma 3.5.13, we know that any fact that is deduced is α-deduced for some recursive ordinal α.) By Lemma 3.5.15, all of those facts are correct. Thus, if there are infinitely many facts deducible from 0, then $\mathcal{O}$ can uniformly compute infinitely many bits of A.

Now, if for every σ that is a finite partial oracle for C, there exists a $\tau \succ_1 \sigma$ that is a finite partial oracle for C which does not deduce any more facts than σ does, then we can build an infinite partial oracle, (C), for C by letting σ_0 be the empty oracle, and at each stage letting σ_{s+1} be an extension of σ_s which does not deduce any additional facts about A. Then $\mathrm{dom}(\varphi^{(C)})$ cannot possibly be infinite, because, if $\varphi^{(C)}(n) = i$, then there is an s such that $\varphi^{\sigma_s}(n) = i$. ($s$ is obtained by letting i be large enough that σ_i includes all of the bits of (C) that were queried in the computation of $\varphi^{(C)}(n)$.) However, in that case, σ_s deduces that $\varphi(n) = i$, and so, by construction, σ_0 deduces that $\varphi(n) = i$. Since there are only finitely many facts that are deduced by the empty oracle, there can only be finitely many n such that $\varphi^{(C)}(n)$ halts. This contradicts the fact that $C \geq_{ii} A$ via φ.

Now, fix a σ that is a finite partial oracle for C such that for any $\tau \succ_1 \sigma$, if τ is also a partial oracle for C, then $\#\mathrm{dom}(f_\tau) > \#\mathrm{dom}(f_\sigma)$.

Consider the set of all $\tau \succ_1 \sigma$ such that f_τ does not incorrectly compute any bits of A. Let X be the set of numbers that are in the domain of f_τ for one of these τ but that are not in the domain of f_σ. Then, one of the following things happens.

1. If there exist infinitely many $\tau \succ_1 \sigma$ such that f_τ incorrectly computes any bits of A, then $A \oplus \mathcal{O} \rightarrow_{T,ii} C$.

   This is because $\mathcal{O}$ can search for 1-extensions of σ which deduce facts that σ does not deduce, and it can figure out what those deduced facts are. Whenever it finds such a 1-extension, A can tell whether or not the additional fact is true. If the additional fact is false, then, by Lemma 3.5.15, it must be because τ is not a partial oracle for C, and in particular, the single bit in $\mathrm{dom}(\varphi^\tau) \setminus \mathrm{dom}(\varphi^\sigma)$ must be incorrect. Thus, $A \oplus \mathcal{O}$ can figure out infinitely many of the bits of C, by ruling out infinitely many incorrect 1-extensions of σ.

2. If there exist finitely many $\tau \succ_1 \sigma$ such that f_τ incorrectly computes any bits of A, and if X is infinite, then $\mathcal{O} \rightarrow_{T,ii} A$.

   This is because $\mathcal{O}$ can compute infinitely many bits of A by a computation which knows σ, knows all of the τ which give incorrect outputs, and which then gives any output given by f_τ for any $\tau \succ_1 \sigma$ other than the τ which give incorrect outputs. Again, for any τ, $\mathcal{O}$ can uniformly enumerate the outputs of f_τ.

3. If there exist finitely many $\tau \succ_1 \sigma$ such that f_τ incorrectly computes any bits of A, and if X is finite, then, by the pigeonhole principle, there must exist some $n \in X$ such that for infinitely many $\tau \succ_1 \sigma$, $f_\tau(n)$ is defined, and is equal to $A(n)$. (Again, by assumption on σ, if $\tau \succ_1 \sigma$ is a partial oracle for C, then $\mathrm{dom}(f_\tau) \supsetneq \mathrm{dom}(f_\sigma)$.)

   This contradicts our definition of deduction, since if there exist infinitely many $\tau \succ_1 \sigma$ such that τ deduces that $\varphi(n) = i$, then σ already deduces that $\varphi(n) = i$, and so n could not have been in X.

This exhausts all the cases, thus, we have that if $C \geq_{ii} A$, then either $\mathcal{O} \rightarrow_{T,ii} A$, or $A \oplus \mathcal{O} \rightarrow_{T,ii} C$.

□

From this, we can immediately conclude Proposition 3.5.9. Then, to conclude Theorem 3.5.7, assuming Proposition 3.5.9, we only need to show that there exist reals satisfying the hypothesis of Proposition 3.5.9.

Lemma 3.5.16. *There exist reals A and B such that $A \oplus \mathcal{O} \not\rightarrow_{T,ii} B$, and $B \oplus \mathcal{O} \not\rightarrow_{T,ii} A$.*

Proof. The construction is a standard construction, and is carried out recursively in $\mathcal{O}'$.

We build A and B by finite approximation, ensuring that for each e, if $\varphi_e^{A\oplus\mathcal{O}}$ has infinite domain, then there exists an n such that $\varphi_e^{A\oplus\mathcal{O}}(n) \neq B(n)$, and likewise that if $\varphi_e^{B\oplus\mathcal{O}}$ has infinite domain, then there exists an n such that $\varphi_e^{B\oplus\mathcal{O}}(n) \neq A(n)$.

At stage $2e$, we have approximations σ and τ to A and B, and we ask whether there is any extension, σ_1 of σ that would cause $\varphi_e^{\sigma_1\oplus\mathcal{O}}(n)$ to halt on some $n > |\tau|$. If there is, then we extend σ to σ_1, and extend τ to some τ_1 so that $\tau_1(n) \neq \varphi_e^{\sigma_1\oplus\mathcal{O}}(n)$. If there isn't, then it is already guaranteed that the domain of $\varphi_e^{A\oplus\mathcal{O}}$ will be finite, and so $\varphi_e^{A\oplus\mathcal{O}}$ will not be an infinite information computation of anything.

At stage $2e+1$, we similarly ensure that $\varphi_e^{B\oplus\mathcal{O}}$ is not an infinite information computation of A.

□

This concludes the proof of Theorem 3.5.7, that there are maximal pairs in the infinite information degrees.

At this point, it would seem that the natural next question is whether there exist maximal infinite information degrees.

Question 8. *Does there exist a real A such that for all B, if $B \geq_{ii} A$, then $B \equiv_{ii} A$?*

Bibliography

[1] George B. Dantzig. "Maximization of a linear function of variables subject to linear inequalities". In: *Activity Analysis of Production and Allocation.* Cowles Commission Monograph No. 13. New York, N. Y.: John Wiley & Sons Inc., 1951, pp. 339–347.

[2] Rod Downey, Carl G. Jockusch Jr., and Paul E. Schupp. "Asymptotic density and computably enumerable sets". submitted for publication.

[3] Santiago Figueira, Joseph S. Miller, and André Nies. "Indifferent sets". In: *J. Logic Comput.* 19.2 (2009), pp. 425–443.

[4] L. G. Hačijan. "A polynomial algorithm in linear programming". In: *Dokl. Akad. Nauk SSSR* 244.5 (1979), pp. 1093–1096.

[5] Gregory Igusa. "Nonexistence of Minimal Pairs for Generic Computability". To appear in J. Symbolic Logic, see also http://arxiv.org/abs/1202.2560.

[6] Carl G. Jockusch Jr. and Paul E. Schupp. "Generic computability, Turing degrees, and asymptotic density". In: *J. Lond. Math. Soc. (2)* 85.2 (2012), pp. 472–490.

[7] Ilya Kapovich and Paul Schupp. "Genericity, the Arzhantseva-Ol′shanskii method and the isomorphism problem for one-relator groups". In: *Math. Ann.* 331.1 (2005), pp. 1–19.

[8] Ilya Kapovich, Paul Schupp, and Vladimir Shpilrain. "Generic properties of Whitehead's algorithm and isomorphism rigidity of random one-relator groups". In: *Pacific J. Math.* 223.1 (2006), pp. 113–140.

[9] Ilya Kapovich et al. "Generic-case complexity, decision problems in group theory, and random walks". In: *J. Algebra* 264.2 (2003), pp. 665–694.

[10] Alexander S. Kechris. *Classical descriptive set theory.* Vol. 156. Graduate Texts in Mathematics. New York: Springer-Verlag, 1995, pp. xviii+402.

[11] Victor Klee and George J. Minty. "How good is the simplex algorithm?" In: *Inequalities, III (Proc. Third Sympos., Univ. California, Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin).* New York: Academic Press, 1972, pp. 159–175.

[12] Leonid A. Levin. "Average case complete problems". In: *SIAM J. Comput.* 15.1 (1986), pp. 285–286.

[13] Gerald E. Sacks. *Higher recursion theory.* Perspectives in Mathematical Logic. Berlin: Springer-Verlag, 1990, pp. xvi+344. ISBN: 3-540-19305-7. DOI: `10.1007/BFb0086109`. URL: `http://dx.doi.org/10.1007/BFb0086109`.

[14] Robert I. Soare. *Recursively enumerable sets and degrees.* Perspectives in Mathematical Logic. A study of computable functions and computably generated sets. Berlin: Springer-Verlag, 1987, pp. xviii+437.

[15] Robert M. Solovay. "Hyperarithmetically encodable sets". In: *Trans. Amer. Math. Soc.* 239 (1978), pp. 99–122.

Appendix A

Coarse Computation

Here, we prove Lemma A.0.17, which says that there exist generically computable sets which are not coarsely computable, which is the second half of Proposition 1.1.13.

For this proof, recall Definitions 1.3.5, and 1.3.6, and Lemma 1.3.7, which establish a means of manipulating the densities of sets by splitting the natural numbers into countably many sets P_i, and then placing "gaps" into those sets. When we build a real with this technique, if a real has countably many gaps of the same "size," then it is not density-1, but if, for every e, the real has at most finitely many gaps of size e, then the real is density-1.

Lemma A.0.17. *There exists a generically computable real that is not coarsely computable.*

Proof. In this proof, we do not explicitly build a real satisfying the lemma. Rather, we explicitly build a generic computation, ψ while ensuring that any real that is generically computed by ψ is not coarsely computable. (Notice that this is, in fact, necessary, since all the reals that are generically computed by ψ differ on a density-0 set, so if one of them is coarsely computable, then they all are, via the same coarse computation.)

For each e, there will be a strategy that ensures that if φ_e is total, then the set of places where it is correct has infinitely many gaps of size e. This strategy will cause $\mathrm{dom}(\psi)$ to have at most one gap of size e, and it will do so in the event that φ_e is not total.

For each e, there will be a strategy S_e, and we will assign countably many P_i to each S_e. Call them $P_{i,e}$.

The eth strategy behaves as follows.

It starts by placing a gap of size e into the domain of ψ at $P_{0,e}$ (and for the other n in $P_{0,e}$, defining $\psi(n) = 0$), and it also starts emulating φ_e to see if it halts on all of $P_{0,e}$.

The intention of this is that if φ_e ever halts on all of $P_{0,e}$, then S_e will fill in the gap that it placed, and make $\psi(n) \neq \varphi_e(n)$ for all n in the last $\frac{1}{2^e}$ of $P_{0,e}$. This will create one gap of size e in $\{n \mid \varphi_e(n) = \psi(n)\}$ without leaving any lasting gap in the domain of ψ.

While S_e is waiting for φ_e to halt on $P_{0,e}$, it slowly fills in the values of ψ on $P_{i,e}$ to halt and output 0. (If after s steps, φ_e has not yet halted on all of $P_{0,e}$, then for $0 < i < s$, if $n \in P_{i,e}$, then $\psi(n) = 0$.)

If φ_e halts on all of $P_{0,e}$, then S_e fills in the gap in the manner previously described, and it selects a new value of i_1 (larger than the stage at which φ_e halted on $P_{0,e}$), and restricts the domain of ψ to have a gap of size e at $P_{i_1,e}$ until it sees φ_e halt on all of $P_{i_1,e}$.

S_e repeats this process for the entire construction, and at the end of the construction, one of two things have happened.

Either it only acted finitely often, in which case there is some i such that φ_e never halted on all of $P_{i,e}$. In this case, φ_e is not a total function, and so it cannot be a coarse computation of anything. Also, in this case, S_e causes the domain of ψ to have exactly one gap of size e (at the last $P_{i,e}$ that it selected.)

Otherwise, S_e acted infinitely often, in which case there are infinitely many values of i such that $\psi(n) \neq \varphi_e(n)$ for n among the last $\frac{1}{2^e}$ of the elements of $P_{i,e}$. (Every time it acts, it creates one more such value of i.) In this case, by Lemma 1.3.7, we know that $\{n \,|\, \psi(n) \neq \varphi_e(n)\}$ is not density-0, and so φ_e cannot possibly be a coarse computation of any real that ψ is a generic computation of.

Now, if we simply run all of the strategies S_e in parallel, and allow each of them to define its respective domain in the manner that it desires, then ψ will be defined everywhere except with, for every e, at most one gap of size e in its domain. Thus, by Lemma 1.3.7 the domain of ψ will be density-1, ψ is defined uniformly recursively on its domain, and for any e, φ_e is not a coarse computation of any set that is generically computed by ψ.

If we let A be any of the sets generically computed by ψ, then A is generically computable, but not coarsely computable.

□

Appendix B

Cone Avoidance

In this section, we prove a theorem of Gandy, Kreisel, and Tait, known as the cone avoidance basis theorem, which says basically that given a recursive tree, T, and a nonrecursive Δ^0_2 real A, $0'$ can be used as an oracle to uniformly compute a path Z through T such that Z does not compute A. (In other words, such that Z avoids the cone above A.)

Theorem B.0.18. *For any infinite recursive tree T, and any nonrecursive Δ^0_2 set A, $0'$ can uniformly (in indices for T and A) compute an infinite path Z through T such that $Z \not\geq_T A$.*

Proof. We will construct Z in stages.

At stage s, we will have an infinite recursive tree $T_s \subseteq T$, and we will narrow it to a new infinite recursive tree $T_{s+1} \subseteq T_s$ such that for every path X though T_{s+1}, φ^X_s is not a computation of A.

As we do this, at every stage, we will ensure that for every length $l < s$, there is exactly one $\sigma \in T_s$ with $|\sigma| = l$. This will ensure that Z is computed by the construction, since we will only need to do s many steps of the construction to compute the first s bits of Z. (Z will be the unique path through the intersection of all of the trees that are built in this construction.)

The actual construction is as follows.

Let $T_0 = T$.

For $s \geq 0$, at stage s, we have a recursive tree T_s, and we do the following.

For each $n \geq 0$, let $T_{s,n}$ be the tree of all σ in T_s such that either $\varphi^\sigma_e(n) \neq A(n)$, or $\varphi^\sigma_e(n)$ does not halt in less than or equal to $|\sigma|$ many steps. $T_{s,n}$ is clearly recursive (For any fixed value of n the value of $A(n)$ can be coded directly in to the computation.)

We now prove three things.

First, if X is a path through $T_{s,n}$, then φ^X_e is not a computation of A. Second, there exists an n such that $T_{s,n}$ is infinite. Third $0'$ can uniformly find such an n.

Let X be a path through $T_{s,n}$. We show that it is not possible that $\varphi^X_e(n) = A(n)$.

To show this, assume that $\varphi^X_e(n) = A(n)$. Let m be the max of the number of steps before $\varphi^X_e(n)$ halts, and the largest number that is queried by φ_e during the computation of

$\varphi_e^X(n)$. Let $\sigma = X \upharpoonright m$. Then $\sigma \notin T_{s,n}$, since $\varphi_e^\sigma(n) = \varphi_e^X(n) = A(n)$, and this computation halts in less than or equal to $|\sigma|$ many steps. Thus, there is an initial segment of X that is not in $T_{s,n}$, so X is not a path through $T_{s,n}$, providing a contradiction.

Now, assume that for every n, $T_{s,n}$ is finite. Then we claim that we can compute A by the following algorithm.

For any fixed value of n, search for a value of m such that for all $\sigma \in T_s$ of length m, $\varphi_e^\sigma(n)$ has halted in less than or equal to m steps, and such that all of those computations give the same output. (Such an m must be found eventually, because any m greater than the height of $T_{s,n}$ is such an m.) Then $n \in A \leftrightarrow \varphi_e^\sigma(n) = 1$ for some $\sigma \in T_s$ of length m. (For any such m that is found, the output must be correct, because once every path through T_s has given an output, then none of them can change their minds in the future, and so if that output is incorrect, then since T_s is infinite, $T_{s,n}$ would be infinite.)

This algorithm always halts, and always gives correct outputs. It only references T_s, which is a single recursive tree, and so it is a recursive algorithm.

Finally, $0'$ can uniformly find such an n because $0'$ can uniformly determine whether or not $n \in A$, and because $0'$ can uniformly ask whether a given recursive binary branching tree is infinite. ($0'$ computes A, and uses knowledge of the bits of A to uniformly determine recursive indices for the trees $T_{s,n}$. For each such tree, it asks the question "Consider the program which looks for a finite height past which $T_{s,n}$ does next extend, and which halts if it finds that height. Does that program ever halt?" When it finds a value of n such that the answer is "no" then that is the n that it is looking for.)

At this point, for each $\sigma \in T_{s,n}$, if $|\sigma| = s$, then $0'$ asks whether the part of $T_{s,n}$ extending σ is infinite. (In practice, there will be at most two such σ's.) For at least one of those values of σ, the answer will be "yes," since otherwise $T_{s,n}$ is finite. Let τ be the leftmost such σ.

Let T_{s+1} be the part of $T_{s,n}$ that goes through τ.

More formally, $\sigma \in T_{s+1} \leftrightarrow (\sigma \in T_{s,n}) \wedge ((\sigma \preceq \tau) \vee (\sigma \succ \tau))$.

Then, for every s, T_{s+1} is obtained uniformly from T_s using $0'$ as an oracle, and for every X that is a path through T_{s+1}, φ_s^X is not a computation of A. So if we let Z be the unique path that is through all of the T_s, then Z is a path through T, Z does not compute A, and Z is obtained uniformly by $0'$ from indices for A and for T.

□

Appendix C

$\boldsymbol{\Pi}^1_1$-Completeness

In this appendix, we define what it means for a set of reals to be $\boldsymbol{\Pi}^1_1$-complete (Boldface $\boldsymbol{\Pi}^1_1$ should not be confused with lightface Π^1_1, which we will define when we discuss hyperarithmetic reals.) For a more comprehensive presentation of the subject, see [10].

Before we do that, however, we will need to state a couple other definitions.

Definition C.0.19. Let S be a subset of the reals 2^ω.

Then we say that S is $\boldsymbol{\Pi}^1_1$ if it can be defined by a Π^1_1 formula with real parameters.

More formally, S is $\boldsymbol{\Pi}^1_1$ if there is a formula Φ of second-order arithmetic, and a parameter $B \in 2^\omega$ such that Φ has three free variables that range over reals, and no quantified variables that range over reals, and such that, for any $A \in 2^\omega$, A is in S if and only if $\forall X \Phi(A, B, X)$.

Definition C.0.20. Let S be a subset of the reals.

Then we say that S is *Borel* if it is $\boldsymbol{\Delta}^1_1$, or in other words, if both S and $2^\omega \setminus S$ are $\boldsymbol{\Pi}^1_1$.

It should be noted that the collection of Borel sets can alternatively be defined as the smallest collection of sets of reals which contains all the open sets of reals, and which is closed under complements, and countable unions and intersections. We will not need this characterization.

Definition C.0.21. Let f be a function $f : 2^\omega \to 2^\omega$.

Then we say that f is Borel if the graph of f is Borel as a subset of $2^\omega \times 2^\omega$. (Here, and in general for the purpose of complexity of sets of reals, $2^\omega \times 2^\omega$ can be thought of as the same thing as 2^ω, using the homeomorphism $\{X, Y\} \mapsto X \oplus Y$ to translate between them.)

Most maps that are defined uniformly, including any sort of uniform reduction, and also the map sending $A \mapsto A'$ are Borel. Also, the composition of Borel maps is Borel.

Definition C.0.22. Let S be a $\boldsymbol{\Pi}^1_1$ set.

Then we say that S is $\boldsymbol{\Pi}^1_1$-*complete* if for every $\boldsymbol{\Pi}^1_1$ set $\widetilde{S} \subseteq 2^\omega$, there exists a Borel function f fuch that $\forall X \in 2^\omega (X \in \widetilde{S} \longleftrightarrow f(X) \in S)$.

The idea behind this is that a question is $\mathbf{\Pi^1_1}$-complete if being able to answer that question allows one to answer any other $\mathbf{\Pi^1_1}$ question in a Borel way. Thus, we determine membership in our other $\mathbf{\Pi^1_1}$ set by mapping its elements over to the first set, and answering the question of membership in the first set.

From the point of view of recursion theory, the most important $\mathbf{\Pi^1_1}$-complete set is the set of well-founded trees on $\omega^{<\omega}$. (Here, $\omega^{<\omega}$ can be thought of as the set of finite sequences of natural numbers. A tree on $\omega^{<\omega}$, like a tree on $2^{<\omega}$, is a subset of $\omega^{<\omega}$ that is closed under initial segments. A tree is *well-founded* if there are no infinite paths through T. A tree on $2^{<\omega}$ is well-founded if and only if it is finite, but a tree on $\omega^{<\omega}$ can have branches of arbitrarily high length while still avoiding having any infinite branches.)

In our proof of Theorem 2.2.1, we create a Borel function which, given a tree, $T \subseteq \omega^{<\omega}$, produces a pair of reals A, and B such that $A \geq_g B$ if and only if T is well-founded. This shows that $\geq_g$ is $\mathbf{\Pi^1_1}$-complete (when thought of as a subset of $2^\omega \times 2^\omega$), since any $\mathbf{\Pi^1_1}$ question can be translated into a question about well-foundedness of trees by a Borel map, and then, in turn, can be translated from that into a question about generic reducibility.

The purpose of showing that a relationship is $\mathbf{\Pi^1_1}$-complete is that it shows that, in some sense, that relationship is as complicated as possible, for a relationship with a Π^1_1 definition. In particular, it shows that the relationship cannot possibly be arithmetically definable, or even Borel definable.

Appendix D

Hyperarithmetic Sets, and Higher Recursion Theory

In this appendix, we define the hyperarithmetic reals, and provide enough background to understand Theorems 2.2.14 and 3.5.7.

Like the set of Borel sets, the set of hyperarithmetic reals can be characterized in either of two ways: as an intersection of larger sets, or a countable union of smaller sets. The first is easier to define, but we will need the second for our proof of Theorem 3.5.7, and so we present both here.

For a more comprehensive presentation of the subject, see [13].

Definition D.0.23. Let $A \in 2^\omega$ be a real.

Then we say that A is Π^1_1 if it can be defined by a Π^1_1 formula.

More formally, A is Π^1_1 if there is a formula Φ of second-order arithmetic such that Φ has one free variable that ranges over reals and one free variable that ranges over natural numbers, and no quantified variables that range over reals, and such that, $n \in A$ if and only if $\forall X \Phi(n, X)$.

Definition D.0.24. Let $A \in 2^\omega$ be a real.

Then we say that A is *hyperarithmetic* if it is Δ^1_1, or in other words, if both A and $\mathbb{N} \setminus A$ are Π^1_1.

(Note that strictly speaking, this is a definition of Δ^1_1, and the fact that it coincides with the hyperarithmetic reals, whose original definition we will present shortly, is a theorem of Kleene.)

For the purpose of Theorem 2.2.14 the only thing that needs to be known is that the hyperarithmetic reals are precisely the reals that can be computed from any sufficiently fast growing function.

Theorem D.0.25. (Solovay) [15]

Let A be a real. Then A is hyperarithmetic if and only if there is a function f, and a Turing functional φ such that for every function g dominating f, φ^g is a computation of A.

(Here, g dominates f if and only if $\forall n$, $g(n) \geq f(n)$.)

We prove here that if A can be computed from a sufficiently fast growing function, then A is Δ^1_1. The converse will be deferred until after we present the original definition of hyperarithmetic reals.

Proof. Let A be a real, and assume that for every function g dominating f, φ^g is a computation of A.

Then, we provide a Δ^1_1 characterization of A as follows.

$n \in A$ if and only if there exists a function f such that for every σ, if σ is a finite approximation to the graph of a function dominating f, then there exists a $\tau \succ \sigma$ which is also a finite approximation to the graph of a function dominating f such that $\varphi^\tau(n) = 1$.

(Note that finite approximations to subsets of $\mathbb{N} \times \mathbb{N}$ can be coded as integers. Thus, the only quantifier over reals in the definition was the existence of an f. In this, we are inherently working with functions in terms of their graphs, so really, this quantifier is saying "there exists an $X \subseteq \mathbb{N} \times \mathbb{N}$ such that for every n, there is exactly one m such that $\langle n, m \rangle \in X$.")

This provides a Σ^1_1 characterization of A. (A Π^1_1 characterization of $\mathbb{N} \setminus A$.)

Also, $n \in A$ if and only if for every function f, there exists a σ which is a finite approximation to the graph of a function dominating f such that $\varphi^\sigma(n) = 1$.

(By assumption, we know that there is an f such that for every g dominating f, $\varphi^g(n) = A(n)$. Thus, if for every f, there is something dominating f that gives a given answer, then that answer must be correct, since, in particular, it is true if the f is the "correct" f.)

This provides a Π^1_1 characterization of A.

Therefore, A is Δ^1_1, and so A is hyperarithmetic.

$\square$

To help motivate the definitions that we will present, we first engage in a mental exercise.

For each n, we can let $0^{(n)}$ be the nth jump of 0. (So $0^{(0)} = 0$, and $0^{(n+1)} = (0^{(n)})'$, the halting set relative to $0^{(n)}$.)

Each time we take the jump, the Turing degree increases. (So $0^{(n+1)} \gneq_T 0^{(n)}$.)

If we want to, however, we can take the uniform join of all of these degrees, and let $0^{(\omega)} = \{\langle n, m \rangle \mid m \in 0^{(n)}\}$. Which is, in some sense, the ωth jump of 0.

We can keep taking jumps, to get $0^{(\omega+1)}$, $0^{(\omega+2)}$, eventually $0^{(\omega \cdot 2)}$, $0^{(\omega^2)}$, $0^{(\omega^3)}$, $0^{(\omega^\omega)}$, etc.

At first glance, it would appear that we should be able to define $0^{(\alpha)}$ for any countable ordinal α, but we eventually reach a problem with the "take a uniform join" step of the operation, since we can only take a uniform join if we have a way of presenting α in a recursive manner. For this reason, $0^{(\alpha)}$ will only be defined for recursive ordinals α.

To make all of this precise, we begin by presenting a notation for discussing recursive ordinals.

In the following definition, $|n| = \alpha$ should be read as "the number n is a notation for the ordinal α."

Definition D.0.26.

- $|1| = 0$.
- *If* $|u| = \alpha$ *then* $|2^u| = \alpha + 1$.
- *Let* a_n *be the nth number enumerated by* φ_e. *Then, if for every* n, $|a_n| = \alpha_n$, *and if, furthermore, for every* n, $\alpha_{n+1} > \alpha_n$, *then* $|3 \cdot 5^e| = \lim_{n\to\infty}\alpha_n$.

Notice that this definition allows for there to be multiple notations for the same ordinal. We will see in Lemma D.0.30 that this does not present a major problem when defining $0^{(\alpha)}$.

Definition D.0.27. *Then, Kleene's* $\mathcal{O}$ *is the set of notations for recursive ordinals: the smallest subset of the natural numbers that has the following properties.*

- $1 \in \mathcal{O}$.
- *If* $u \in \mathcal{O}$, *then* $2^u \in \mathcal{O}$.
- *If, for every* n, *the nth number enumerated by* φ_e, a_n *is in* $\mathcal{O}$, *and if for every* n, $|a_{n+1}| > |a_n|$, *then* $3 \cdot 5^e \in \mathcal{O}$.

Definition D.0.28. α *is a recursive ordinal if* $\exists u \in \mathcal{O}$, $|u| = \alpha$.
ω_1^{CK} *is the smallest nonrecursive ordinal.*

We are now ready to define the H-sets, which will be our formalization of how to take an ordinal number of jumps of 0.

Definition D.0.29. *Let* $u \in \mathcal{O}$. *Then* H_u *is the real defined by induction in the following manner.*

- $H_1 = 0$
- $H_{2^u} = H'_u$
- $H_{3\cdot 5^e}$ *is the uniform join of the* H_{a_n} *where* a_n *is the nth number enumerated by* φ_e.

  More formally, let a_n *be the nth number enumerated by* φ_e. *Then* $H_{3\cdot 5^e} = \{\langle x, n\rangle \mid x \in H_{a_n}\}$.

This definition allows us to make precise what we mean when we refer to $0^{(\alpha)}$.

Lemma D.0.30. (Spector)
Let $u, v \in \mathcal{O}$. *Then* $H_u \equiv_T H_v$ *if and only if* $|u| = |v|$.
Thus, for any recursive ordinal α, *we may define* $0^{(\alpha)}$ *by fixing any* $u \in \mathcal{O}$ *such that* $|u| = \alpha$, *and letting* $0^{(\alpha)} = H_u$. *This definition is only well-defined up to Turing degree, but that is sufficient to determine what computes* 0^α, *and what is computable from* $0^{(\alpha)}$.

We now present the original definition of hyperarithmetic.

Definition D.0.31. Let $A \in 2^\omega$ be a real.

Then A is *Hyperarithmetic* if and only if there is some recursive ordinal α such that $0^{(\alpha)} \geq_T A$.

Theorem D.0.32. (Kleene)

Definitions D.0.24 and D.0.31 are equivalent.

The proof is omitted, but can be found in [13].

We now prove Theorem D.0.25 by first proving that every H-set has a modulus.

Lemma D.0.33. *Let $u \in \mathcal{O}$. Then there exists an f and an e such that for any g dominating f, φ_e^g is a computation of H_u. Furthermore, an index for e can be computed uniformly from u.*

Proof. We prove the statement by induction on $|u|$.

If $u = 1$, so $H_u = 0$, which can be uniformly computed from any function.

If $u = 2^v$, then by induction, fix some f_0, e_0 such that for any g dominating f_0, $\varphi_{e_0}^g$ is a computation of H_v. Let f_1 be the smallest function with the property that if $\varphi_e^{H_v}(e)$ halts, then $\varphi_e^{H_v}(e)$ halts in less than or equal to $f_1(e)$ many steps.

Let $f(n) = \max(f_0(n), f_1(n))$.

Then, from any g dominating f, we can compute H_u as follows.

Compute H_v using $\varphi_{e_0}^g$. Then use our computed values for H_v to run approximations of $\varphi_e^{H_v}(e)$. Then, $e \in H_u$ if and only if $\varphi_e^{H_v}(e)$ halts in less than or equal to $g(e)$ many steps. This construction is uniform in e_0, so if we could uniformly compute e_0 from v, then we can uniformly compute an index for this construction from u.

If $u = 3 \cdot 5^k$, then, for each n, let u_i be the ith number enumerated by φ_k. By induction, fix f_i, e_i such that for any g dominating f_i, $\varphi_{e_i}^g$ is a computation of H_{u_i}.

Then, let $f(n) = \max_{m \leq n}\{f_m(n-m)\}$.

Then, let g dominate f. Let $g_i(n) = g(n+i)$. Note then that g_i dominates f_i, so to compute whether or not $\langle x, i\rangle \in H_u$, we use $\varphi_{e_i}^{g_i}$ to compute whether or not $x \in H_{u_i}$.

The u_i can be computed uniformly from k (since we are assuming that $u \in \mathcal{O}$, and so, in particular, φ_k will eventually enumerate infinitely many things), the e_i can be computed uniformly from the u_i by induction, and the rest of the computation is uniform in the e_i, so an index for the computation can be recovered uniformly from u.

□

We can now easily prove the forward direction of Theorem D.0.25.

Corollary D.0.34. (Solovay) [15]

Let A be a hyperarithmetic real. There is a function f, and a Turing functional φ such that for every function g dominating f, φ^g is a computation of A.

Proof. By Definition D.0.31, fix a u such that $H_u \geq_T A$. By Lemma D.0.33, choose f_0 such that any g dominating f_0 can be used uniformly to compute H_u. Fix i, j such that $\varphi_i^{H_u}$ is a computation of A, and such that for any g dominating f_0, φ_j^g is a computation of H_u.

Then, let f be f_0. To compute A from any g dominating f, use φ_j to compute H_u, and then use the computed values of H_u with φ_i to compute A.

□

We conclude this appendix by mentioning, without proof, various results concerning recursive ordinals and H-sets.

Lemma D.0.35. *If $a \in \mathcal{O}$, then H_a can uniformly enumerate the $b \in \mathcal{O}$ such that $|a| > |b|$. Furthermore, for each such b, H_a can uniformly compute H_b.*

Lemma D.0.36. *$\mathcal{O}$ can uniformly compute H_a for any $a \in \mathcal{O}$.*

Definition D.0.37. Let $\Gamma : 2^\omega \to 2^\omega$ be a map from reals to reals. Then *the set inductively defined by* Γ is defined as follows.

For any ordinal α, we define Γ_α by induction.

Γ_0 is the empty set.

$\Gamma_{\alpha+1} = \Gamma(\Gamma_\alpha) \cup \Gamma_\alpha$.

If λ is a limit, then $\Gamma_\lambda = \bigcup_{\alpha<\lambda} \Gamma_\alpha$.

Then, Γ_∞, the set inductively defined by Γ is given by $\Gamma_\infty = \bigcup_{\alpha \in \mathrm{ORD}} \Gamma_\alpha$.

The idea behind this is that Γ is a *closure condition,* and given any real X, $\Gamma(X)$ tells you certain new elements need to be added to that set in order for it to be closed under Γ. To reach a set that is actually closed under Γ, we might need to apply Γ a transfinite number of times.

Observation D.0.38. *If $\alpha < \beta$, then $\Gamma_\alpha \subseteq \Gamma_\beta \subseteq \mathbb{N}$. Also, if $\Gamma_\alpha = \Gamma_\beta$, then for all $\gamma > \alpha$, $\Gamma_\gamma = \Gamma_\beta$. Thus, since new elements can only be added to Γ_∞ a countable number of times, $\Gamma_\infty = \Gamma_{\omega_1}$.*

In particular, since there exists an ordinal, α, such that $\Gamma_\alpha = \Gamma_\infty$, there must be a least such ordinal.

Definition D.0.39. $|\Gamma|$, *is the least ordinal, α, such that $\Gamma_\alpha = \Gamma_\infty$.*

Definition D.0.40. A closure condition, Γ, is *monotonic* if it satisfies the property that for any reals X, and Y, if $X \subseteq Y$, then $\Gamma(X) \subseteq \Gamma(Y)$.

Lemma D.0.41. (Spector)

If Γ is monotonic, and Γ is Π^1_1, then $|\Gamma| < \omega_1^{CK}$, and Γ_∞ is Π^1_1.

(Here, a map on reals is Π^1_1 if there is a formula Φ, of second order arithmetic, with three free variables ranging over reals, and no quantified variables that range over reals, such that for each X, there is a unique Y such that $\forall Z \Phi(X, Y, Z)$, and if, for each X, the unique Y is $\Gamma(X)$.)

The basic idea of Lemma D.0.41 is that it provides a way of concluding, for almost any set that is built in stages, that there is a stage before ω_1^{CK} at which the set is finished being built.